Data General

Programmer's Reference Manual

NOVA LINE COMPUTERS

015-000023-01

Programmer's Reference Manual

NOVA LINE COMPUTERS

015-000023-01

Ordering No. 015-000023

Printed in the United States of America
Rev. 01, March 1975

NOTICE

TABLE OF CONTENTS

SECTION 1

THE NOVA LINE COMPUTERS

SECTION 2

INTERNAL STRUCTURE

TABLE OF CONTENTS (Continued)

SECTION 3

INSTRUCTION SETS

TABLE OF CONTENTS (Continued)

SECTION 4
INPUT/OUTPUT

SECTION 5
PROCESSOR OPTIONS

TABLE OF CONTENTS (Continued)

SECTION 5 (Continued)

PROCESSOR OPTIONS

TABLE OF CONTENTS (Continued)

SECTION 5 (Continued)

PROCESSOR OPTIONS

TABLE OF CONTENTS (Continued)

SECTION 6

FRONT PANEL

APPENDICES

SECTION I

THE NOVA LINE COMPUTERS

INTRODUCTION

The Data General Corporation NOVA® line of computers are general purpose, four-accumulator, stored-program computers, with a word length of 16 bits. The maximum amount of main memory is 32,768 16-bit words. For the NOVA 830 and NOVA 840 computers with the MMPU feature, the maximum amount of main memory is 131,072 16-bit words. The accumulators are also 16 bits in length and are used for arithmetic and logical operations. Furthermore, two of the accumulators can be used as index registers. Memory can be addressed either directly or by using indirect addresses. Chains of indirect addresses can be of any length. A direct memory access (DMA) data channel is provided to enable rapid data transfer between main memory and peripheral devices. The flexible design of the NOVA line of computers allows the convenient implementation of applications in all sectors of the data processing field.

The standard instruction set contains instructions that perform fixed point arithmetic and logical operations between accumulators, transfer of operands between accumulators and main memory, transfer of program control, and input/output (I/O) operations. Options are available that add instructions to this set. These additional instructions perform such operations as multiply/divide, floating point calculations, memory allocation and protection, and memory management and protection.

The NOVA line of computers is made up of the NOVA computer, the SUPERNOVA® computer, the NOVA 1200 series, the NOVA 800 series, and the NOVA 2 series. The NOVA 1200 series consists of the NOVA 1200 computer, the NOVA 1210 computer, the NOVA 1220 computer, and the NOVA 1200 Jumbo computer. The NOVA 800 series consists of the NOVA 800 computer, the NOVA 820 computer, the NOVA 800 Jumbo computer, the NOVA 830 computer, and the NOVA 840 computer. The NOVA 2 series consists of the NOVA 2/4 computer and the NOVA 2/10 computer. While these computers differ in specifics such as processing speed, they all share the same general architecture. This means that, in general, software is compatible across the entire line. To a somewhat lesser degree, hardware is also compatible across the line. The features of the NOVA line are summarized below.

Efficient Basic Instruction Set

The basic instruction set for the NOVA line of computers contains instructions that perform fixed point arithmetic and logical operations between accumulators, transfer of operands between accumulators and main memory, transfer of program control, and I/O operations. All instructions are one 16-bit word in length. The arithmetic and logical instructions have the capability to perform, in one instruction, the following sequence: perform an operation, shift the result one bit left or right, test the result of the shift, and then conditionally skip the next instruction depending upon the outcome of the test. In addition, it is possible to perform this entire sequence without affecting either of the operands. This means that complicated numerical manipulation and testing can be performed using a small number of instructions.

Multiply/Divide

The multiply/divide feature allows the multiplication and division of operands to be performed quickly, without resorting to time-consuming software routines. Two 16-bit fixed point operands can be multiplied together to yield a 32-bit fixed point result. A 16-bit fixed point operand can be divided into a 32-bit fixed point operand to yield a 16-bit fixed point quotient and a 16-bit fixed point remainder.

Floating Point

The floating point feature allows the manipulation of both single precision (32 bits) and double precision (64 bits) floating point numbers. Single precision gives 6-7 significant decimal digits while double precision gives 13-15 significant decimal digits. The decimal range of a floating point number is approximately 5.4x10-79 to 7.2x10+75 in either precision.

The floating point feature contains two 64-bit floating point accumulators. Floating point calculations can take place between these two accumulators or between one of the accumulators and operands in main memory.

Memory Allocation and Memory Management

The memory allocation and protection (MAP) feature and the memory management and protection unit (MMPU) feature perform logical-to-physical address translation and memory protection. In addition, the MMPU feature allows the processor to address up to 131,072 16-bit words of main memory.

The MAP feature allows the allocation of memory to a user in blocks of 4096 words. Up to 8 such blocks may be allocated to a user. The MMPU feature allows the allocation of memory to a user in blocks of 1024 words and up to 32 such blocks may be allocated to a user. In both cases, a user is prohibited from accessing those blocks of memory not allocated to him, thus protecting a user's area of memory from unauthorized access. Both features allow areas of memory to be write-protected and areas of memory to be allocated to more than one user, thus allowing the sharing of data and procedure areas. The blocks of memory allocated to a user do not have to be contiguous.

The address translation function which correlates a logical address to the corresponding allocated physical memory address is called an "address map". The MAP feature is capable of holding the mat for one user at a time and memory references from the data channel are not mapped. The MMPU feature also holds only one user map at a time, but it has the capability of simultaneously mapping memory references for the data channel.

In addition to translating addresses, these two features also perform various protection functions. A user is allowed to access only those blocks of memory allocated to him. This ensures that a user does not reach out of his own areas of memory for either instructions or data. Blocks of memory allocated to a user may be write-protected so that the user may not modify them. This allows blocks of memory containing constants or non-self-modifying procedures to be shared between users. The MAP feature detects and inhibits indirection chains that go deeper than two levels. The MMPU feature detects and inhibits indirection chains that go deeper than 16 levels. In both cases, this protects the system from becoming disabled by an indirection loop. While the MAP feature provides total I/O device protection because the user is not allowed to issue I/O instructions, the MMPU allows devices to be declared accessible or inaccessible to a user on an individual device code basis. This allows any device to be controlled by the operating system or dedicated to a user, depending upon user requirements.

The MAP feature is available on the SUPERNOVA computer, the NOVA 800 computer, and the NOVA 820 computer. The MMPU feature is available on the NOVA 830 computer, and the NOVA 840 computer.

Memory

Memory is available in many forms for the different members of the NOVA line. For the NOVA computer, core memory is available in modules of 2, 4, and 8K 16-bit words. For the SUPERNOVA computer, memory is available in both core and semiconductor forms. Core memory is available in modules of both 4 and 8K 16-bit words. Semiconductor memory is available in both read/write and read-only forms in modules of 256, 512, and 1024 16-bit words. For the NOVA 1200 series of computers, both core and semiconductor memory is available. Core memory is available in modules of 4, 8, and 16K 16-bit words. Semiconductor memory is available in both read/write and read-only forms in modules of 256, 512, and 1024 16-bit words. For the NOVA 800 and 820 computers, core memory is available in modules of 4 and 8K 16-bit words. For the NOVA 830 computer, core memory is available in modules of 16K 16-bit words. For the NOVA 840 computer, core memory is available in modules of 8K 16-bit words. For the NOVA 2 series of computers, core memory is available in modules of 4, 8, and 16K 16-bit words.

Power Fail/Auto Restart

The power fail/auto restart feature of the NOVA line provides a "fail-soft" capability in the event of unexpected power loss. In the event of power failure, there is a delay of one to two milliseconds before the processor shuts down. The power fail portion of the feature senses the imminent loss of power and interrupts the processor. The interrupt service routine can then use this delay to store the contents of the accumulators, the program restart address, and other information that will be needed to restart the system. One to two milliseconds is enough time to execute 200 to 1500 instructions depending on the processor, so there is more than enough time to perform the power fail routine.

When power is restored, the action taken by the auto-restart portion of the feature depends upon the position of the power switch on the front panel. If the switch is in the "on" position, the processor remains stopped after power is restored. If the switch is in the "lock" position, then 50 milliseconds after power is restored, the processor executes the instruction contained in the first location of main memory, restarting the interrupted system.

Real-Time Clock

The real-time clock feature of the NOVA line computers generates a sequence of pulses that is independent of the timing of the processor. The clock will interrupt the system at one of four program-selectable frequencies. The frequencies are: ac line frequency, 10Hz, 100Hz, and 1000Hz.

Input/Output Bus

The input/output (I/O) bus is that portion of the computer that carries commands and data between the central processor and various peripheral devices connected to it. The bus is made up of a six-line device selection network, interrupt circuitry, command circuitry, and sixteen data lines.

Device Addressability

Each I/O device in a NOVA line computer system is connected to the six-line device selection network in such a way that each device will only respond to commands that contain its own device code. The fact that the selection network is made up of six lines gives $2^6 = 64$ unique device codes. Two of these codes are reserved for specific functions, but there are still 62 device codes available for use with I/O devices.

Interrupt Capability

The interrupt circuitry contained in the I/O bus provides the capability for any I/O device to interrupt the system when that device requires service. When a device requests an interrupt, the processor automatically transfers program control to the main interrupt service routine. This routine can either poll all the I/O devices in the system to find out which one initiated the interrupt or the routine can use a special instruction to identify the source of the interrupt.

The interrupt circuitry of the NOVA line also contains the capability to implement up to sixteen levels of priority interrupts. This is done with a 16-bit priority mask. Each level of device priority is associated with a bit in this mask. In order to suppress interrupts from any priority level, the corresponding bit in the mask is set to 1.

Data Channel

Handling data transfers between external devices and memory under program control requires an interrupt plus the execution of several instructions for each word transferred. To allow greater transfer rates, the I/O bus contains circuitry for a direct memory access (DMA) data channel through which a device, at its own request, can gain direct access to main memory using a minimum of processor time. At the maximum transfer rate, the data channel effectively stops the processor, but at lower rates, processing continues while data is being transferred.

Ease of Interfacing

Due to the straightforward logic and general design of the NOVA line I/O bus, customer-provided or customer-designed I/O devices may be easily interfaced to a NOVA line computer system. Information on how to interface to the NOVA line may be found in "The Interface Manual" (DGC 015-000031).

Input/Output Devices

A comprehensive array of I/O devices is available from Data General for the NOVA line. This wide choice of devices, ranging from teletypewriters to line printers to video displays for man-machine interaction; and from paper tape to magnetic tape to fixed and moving-head discs for data storage allows a wide spectrum of possible configurations. Also available are various multiplexors and telecommunications adapters including an IBM 360/370 interface.

Software

The NOVA line is fully supported by proven Data General software. Because all members of the NOVA line are program compatible with each other, it is possible to create a computer system that can be easily altered or upgraded as the need arises.

Languages

In addition to an assembler and a macro-assembler, there are powerful higher-level language processors available for use with the NOVA line. Language processors such as ALGOL, EXTENDED BASIC, FORTRAN IV, and FORTRAN 5 can be used to ease the job of implementing application systems.

Operating Systems

There is a wide array of operating systems available for the NOVA line. These range from the Stand-alone Operating System (SOS) to the Real-Time Operating System (RTOS) to the Real-Time Disc Operating System (FDOS), to the Mapped Real-Time Disc Operating System (MRDOS). SOS, RTOS, and RDOS software are designed for the small to medium-size systems, while MRDOS software is designed for the large system and gives full software support for the Memory Management and Protection Unit.

Conclusion

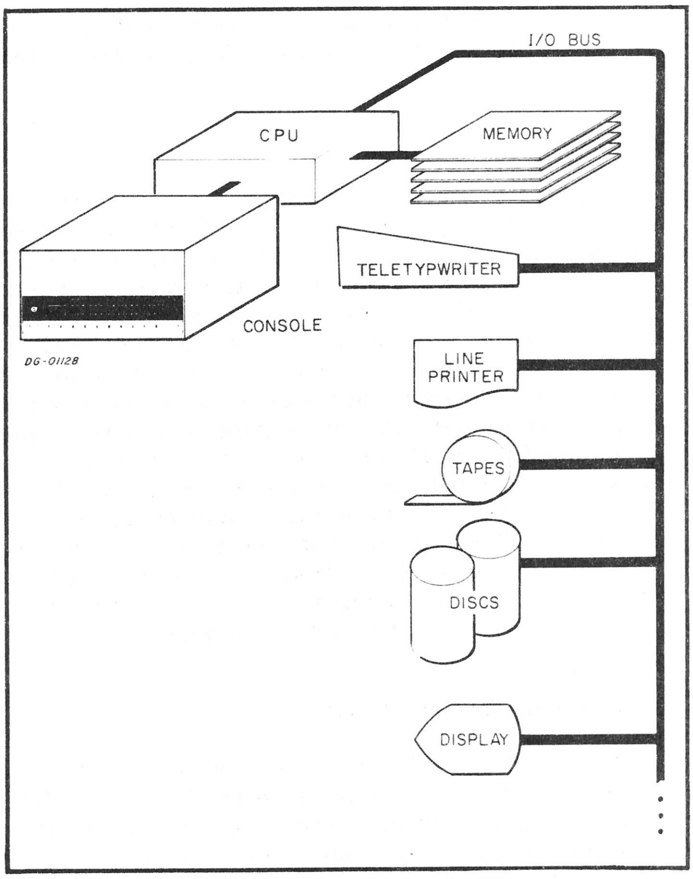
The internal features, software, and I/O devices available with the NOVA line of computers ensure that they will easily meet the continually changing needs of the data processing industry.

This page intentionally left blank.

SECTION 2
INTERNAL STRUCTURE

INTRODUCTION

The basic structure of a NOVA line data processing system consists of a central processing unit (CPU), some amount of main memory, the I/O bus, the I/O devices connected to the I/O bus, and a console which is on the front panel of the main computer chassis.



Due to the general-purpose design of the NOVA line, the type, size, and number of memory modules and I/O devices have no effect upon the internal logical structure of the CPU. This chapter deals with the addressing of information and the logical representation of information within the CPU, and is unaffected by those portions of the system outside the CPU.

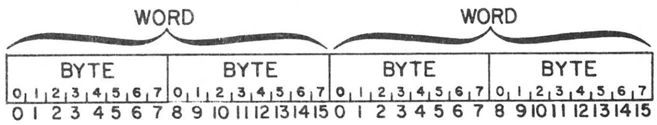
INFORMATION FORMATS

The basic piece of information within the processor is the binary digit, or "bit". A bit is capable of representing only two quantities, 0 and 1. However, a bit cannot represent both these values at the same time. At any one point in time, a bit can either represent a 0 or a 1, never both.

The normal unit of information within the CPU is the "byte". A byte is made up of 8 bits. Because each bit is capable of representing two quantities, a byte is capable of representing $2^8 = 256$ different quantities. Two bytes may be combined to produce a 16-bit unit called a "word". A word can represent $2^{16} = 65,536$ different quantities. I/O devices transfer information in units of bits, bytes, words, or multiples of words called "records", depending upon the device.

Bit Numbering

In order to avoid confusion when talking about the information contained in bytes and words, the bits that make up these units of information are numbered from left to right, with the leftmost (high-order) bit always numbered bit 0. The numbering extends to the right and is always carried out in the decimal number system. The rightmost (low-order) bit in a byte is bit 7. The rightmost bit in a word is bit 15.



Octal Representation

Because talking about the binary data contained in bytes and words would quickly become awkward and confusing if each bit were described, the octal representation of binary information will be used in this manual. To convert a piece of binary information to its octal representation, the bits in the quantity are separated into groups of three bits each, starting from the right and proceeding to the left. If the number of bits to be represented is not evenly divisible into groups of three, the leftmost group will contain one or two bits. Each group of bits can now be represented by one of eight different symbols. The digits 0-7 are used to represent the quantities 0-7. Each encoded digit is called an octal digit. Because each group of bits can contain any one of 8 values, this representation is sometimes called "base 8" representation.

Another way to represent binary information is the hexadecimal or "hex" representation. In hexadecimal, the bits in the quantity are separated into groups of four bits each and each group can be represented by one of 16 different symbols. The digits 0-9 are used to represent the quantities 0-9. The letters A-F are used to represent the quantities 10-15. Because each group of bits can contain any one of 16 values, this representation is sometimes called "base 16" representation.

The following table gives the correspondence between the various representations.

DECIMAL	BINARY	HEX	BINARY	OCTAL
0	0000	0	000	0
1	0001	1	001	1
2	0010	2	010	2
3	0011	3	011	3
4	0100	4	100	4
5	0101	5	101	5
6	0110	6	110	6
7	0111	7	111	7
8	1000	8	1 100	10
9	1001	9	1 001	11
10	1010	A	1 010	12
11	1011	B	1 011	13
12	1100	C	1 100	14
13	1101	D	1 101	15
14	1110	E	1 110	16
15	1111	F	1 111	17

Our normal decimal numbering system is sometimes called "base 10" representation. Because it is sometimes possible to confuse numbers written in hex or octal with those written in decimal, a subscript denoting the base will be used in cases where confusion might occur. The following examples illustrate this convention.

$$64_{10} = 40_{16} = 100_8$$

$$87_{10} = 57_{16} = 127_8$$

$$63_{10} = 3F_{16} = 77_8$$

In the last example, it is obvious that 3F is a number written in hex, but the subscript is included to erase any possible doubts.

Conversion tables for hex to decimal and octal to decimal are contained in Appendix B of this manual.

Character Codes

Within the processor, all information is represented by binary quantities. The CPU does not recognize certain bit combinations as characters and certain other bit combinations as numbers. Sooner or later, however, this information must be transferred outside the computer in some form easily understood by humans. For this reason, some standard correspondence must be made between certain bit combinations and printable symbols. The code used to implement this correspondence in I/O devices available with the NOVA line is called the American Standard Code for Information Interchange (ASCII). This code can represent 95 printable symbols plus 33 control functions. A complete table of the codes and their corresponding characters can be found in Appendix C of this manual.

Information Representation

Even though the CPU does not intrinsically recognize one information type from another, the different instructions in the instruction set expect that the information to be operated on will be in a specific format. In general, there are four different, basic information formats. They are integers, floating point numbers, logical quantities, and decimal numbers.

Integers

Integers can be represented as either signed or unsigned numbers and carried in either single or multiple precision. Single precision integers are two bytes long, while multiple precision integers are four or more bytes long. Unsigned integers use all the available bits to represent the magnitude of the number. A single two-byte word can represent any unsigned number in the inclusive range 0 to 65,535. Two words taken together as an unsigned, double precision integer can represent any number in the inclusive range 0 to 4,294,967,295.

For signed operations, the two's complement numbering system is used. In this system, the leftmost or high-order bit is used as a sign bit. If the sign bit is 0, the number is positive and the remainder of the bits in the number represent the magnitude of the number as described above. If the sign bit is 1, the number is negative and the remainder of the bits represent the two's complement of the magnitude of the number.

To create the negative of a number in the two's complement scheme, complement all the bits of the number including the sign bit. After the complementing process is finished, add 1 to the rightmost or low-order bit. If the two's complement of a negative number is formed, the result will be the corresponding positive number. There is only one representation for zero in two's complement arithmetic: it is the number with all bits zero. Forming the two's complement of zero will produce a carry out of the high-order bit and leave the number with all bits zero.

Examples:

To form the negative of 4:

4 =	0	000	000	000	000	100
complement =	1	111	111	111	111	011
add 1 +						1
-4 =	1	111	111	111	111	100

To form the negative of 1715_8:

1715_8 =	0	000	001	111	001	101
complement =	1	111	110	000	110	010
add 1 +						1
-1715_8 =	1	111	110	000	110	011

To form the negative of -1715_8:

-1715_8 =	1	111	110	000	110	011
complement =	0	000	001	111	001	100
add 1 +						1
1715_8 =	0	000	001	111	001	101

To form the negative of 0:

0 =	0	000	000	000	000	000
complement =	1	111	111	111	111	111
add 1 +						1
0 =	0	000	000	000	000	000

Note that 0 is a positive number, i.e., its sign bit is 0.

Because the two's complement scheme has only one representation for 0, there is always one more negative number than there are non-negative numbers. The most negative number is a number with a 1 in the sign bit and all other bits 0. The positive value of this number can not be represented in the same number of bits as used to represent the negative number.

A single two-byte word can represent any signed number in the inclusive range -32,768 to +32,767. Two words taken together as a signed, double precision integer can represent any number in the inclusive range -2,147,483,648 to +2,147,483,647.

It is a property of numbers using the two's complement scheme that addition and subtraction of signed numbers are identical to addition and subtraction of unsigned numbers. The CPU just treats the sign bit as the most significant magnitude bit.

Floating Point

The floating point feature of the NOVA line allows operations on signed numbers having a much larger range than those normally represented as integers. It would take a 16-word multiple precision integer to represent the range of a NOVA line floating point number. Since floating point numbers occupy either two words for single precision or four words for double precision, and the floating point feature is much faster than multiple precision integer software routines, floating point arithmetic is used when numbers having a large range must be manipulated.

A floating point number is made up of three parts: the sign, the exponent, and the mantissa. The value of a floating point number is defined to be:

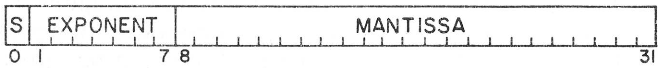
(MANTISSA) X (16 RAISED TO THE TRUE VALUE OF THE EXPONENT FIELD)

The number is signed according to the value of the sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative.

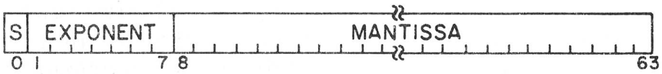
Floating point numbers are represented internally by either 32 bits (single precision) or 64 bits (double precision).

The formats are shown below:

Single Precision



Double Precision



Bit zero is the sign bit: 0 for positive, 1 for negative.

Bits 1-7 contain the exponent. This is the power to which 16 must be raised in order to give the correct value to the number. So that the exponent field may accommodate a large range, "Excess 64" representation is used. This means that the value in the exponent field is 64 greater than the true value of the exponent. If the exponent field is zero, the true value of the exponent is -64. If the exponent field is 64, the true value of the exponent is 0. If the exponent field is 127, the true value of the exponent is 63.

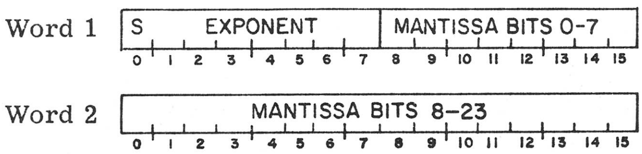
Bits 8-31 for single precision and bits 8-63 for double precision contain the mantissa. This means that bit 8 of the floating point number is bit 0 of the mantissa. The mantissa is always a positive fraction greater than or equal to 1/16 and less than 1. The "binary point" can be thought of as being just to the left of bit 8. Continuing this concept then, bit 8 represents the value 1/2, bit 9 represents the value 1/4, bit 10 represents the value 1/8, and so on.

In order to keep the mantissa in the range of 1/16 to 1, the results of floating point arithmetic are "normalized". Normalization is the process whereby the mantissa is shifted left one hex digit at a time until the high-order four bits represent a nonzero quantity. For every hex digit shifted, the exponent is decreased by one. Since the mantissa is shifted four bits at a time, it is possible for the high-order three bits of a normalized mantissa to be zero.

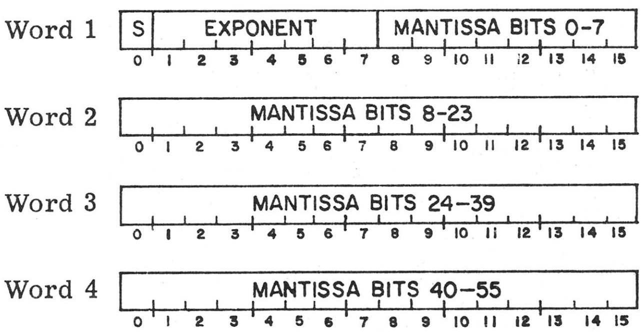
Zero is represented by a floating point number with all bits zero. This is true for both single and double precision. This is known as "true zero". When a calculation results in a zero mantissa, the floating point processor automatically converts the number to a true zero. Note that true zero is positive. It is not possible to obtain negative zero as the result of a calculation.

Floating point operands in memory are represented by two words for single precision and by four words for double precision. The formats are shown below:

Single Precision



Double Precision



Logical Quanities

Logical operations in the NOVA line can be performed upon individual bits, bytes, or words. When using the logical operations, quantities operated on are treated as unstructured binary quantities. The number of bits, bytes, or words operated upon depends on the particular instruction.

Decimal Numbers

Decimal numbers may be represented internally in two ways, character decimal and packed decimal. In character decimal, the number is made up of a string of ASCII characters and the sign, if present, may appear in one of four places. The sign of the number may be indicated by a leading or trailing byte which contains the ASCII code for plus ($2B_{16}$) or minus ($2D_{16}$). Alternatively, either the high-order digit or the low-order digit of the number may indicate the sign in addition to carrying a digit of the number. The table below gives the correspondence between certain ASCII characters and the sign and digit values that they carry.

SIGN VALUE	DIGIT VALUE	ASCII CHARACTER	HEX CODE
+	0	{	7B
+	1	A	41
+	2	B	42
+	3	C	43
+	4	D	44
+	5	E	45
+	6	F	46
+	7	G	47
+	8	H	48
+	9	I	49
-	0	}	7D
-	1	J	4A
-	2	K	4B
-	3	L	4C
-	4	M	4D
-	5	N	4E
-	6	O	4F
-	7	P	50
-	8	Q	51
-	9	R	52

The digits that are not carrying the sign must be valid ASCII characters for the digits 0-9 (30_{16}-39_{16}).

Examples:

In the following examples, the hex value of a byte is shown inside the box; the corresponding ASCII character is shown beneath the box.

+2,048 (leading sign)	2B	32	30	34	38
	+	2	0	4	8
-1,756 (trailing sign)	31	37	35	36	2D
	1	7	5	6	-
+1,850 (high-order sign)	41	38	35	30	
	A	8	5	0	
-3,970 (low-order sign)	33	39	37	7D	
	3	9	7	}	

For packed decimal, each digit of the decimal number occupies one hex digit. The sign is specified by a trailing hex digit. The number must start and end on a byte boundary. In other words, the number cannot start or end halfway through a byte. This means that a packed decimal number will always consist of an odd number of digits followed by the sign. The sign must be either C_{16} for plus or D_{16} for minus. The only valid codes for digits are 0-9_{16}.

Examples:

In the following examples, the hex value of a digit is shown within the box; the corresponding decimal digit is shown beneath the box.

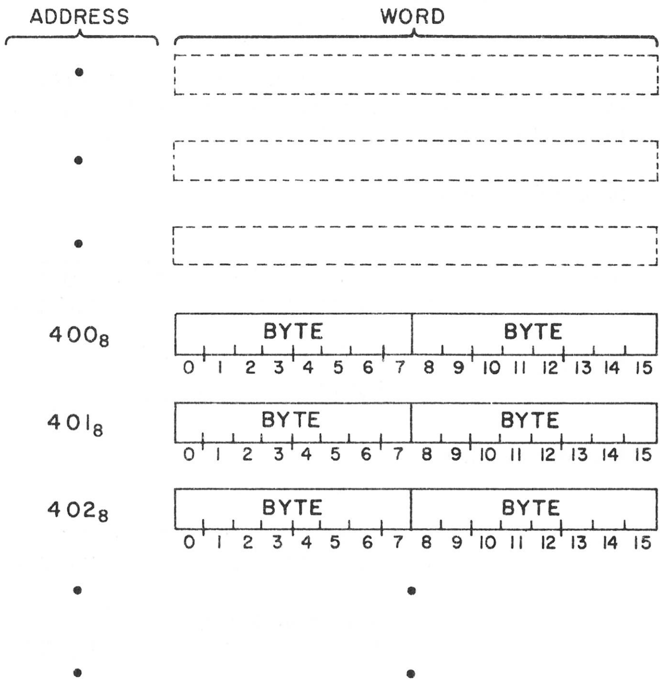
	Byte		Byte		Byte	
+ 2,048	0	2	0	4	8	C
	0	2	0	4	8	+
+32,456	3	2	4	5	6	C
	3	2	4	5	6	+
- 1,756	0	1	7	5	6	D
	0	1	7	5	6	-
-25,989	2	5	9	8	9	D
	2	5	9	8	9	-

INFORMATION ADDRESSING

The information formats described in the preceding section give a way of representing different types of data with the CPU. Operations cannot be performed upon these data types, however, unless they can be addressed by the CPU. The address of a piece of information is its location in main memory. Once the CPU knows the address of a piece of information, the desired operation can be performed.

Word Addressing

Main memory is partitioned into 2-byte words, and each word has an address. The first word in memory has the address 0. The next word has the address 1, the next word has the address 2, and so on. Word addressing is used to address integers, floating point numbers, and logical quantities that are formatted in units of words.



DG-00538

Effective Address Calculation

There are six instructions in the NOVA line instruction set that directly reference memory using word addressing. These instructions use eleven bits in the instruction to define the address of the desired word. These eleven bits do not directly specify the address, but are used in a calculation which results in the address of the desired word. The resultant address is called the "effective address" or "E", and the calculation is called the "effective address calculation".

The eleven bits in an instruction that are used in the effective address calculation, are bits 5-15. Their format is shown below

					@	INDEX		DISPLACEMENT							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bit 5 is called the "indirect bit", bits 6 and 7 are called the "index bits", and bits 8-15 are called the "displacement bits".

If the index bits are 00, the displacement is used as an unsigned 8-bit number to address one of the first 256_{10} words in memory. This is called "page zero addressing" and this first block of 256 words is known as "page zero".

If the index bits are 01, the displacement is treated as a signed, two's complement number, which is added to the address of the instruction to produce a memory address. This is called "relative addressing". By relative addressing, any instruction which uses the effective address calculation can directly address any word in storage whose address is in the range -128_{10} to $+127_{10}$ from the instruction.

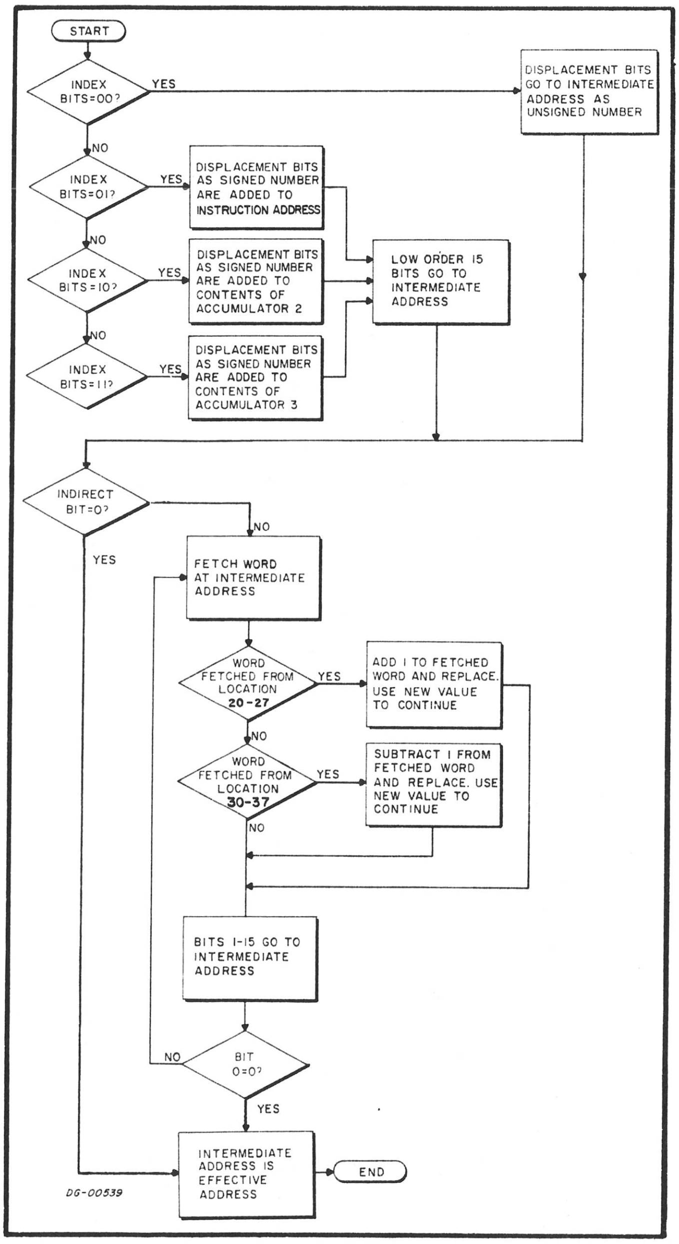
If the index bits are 10, accumulator 2 is used as an index register. If the index bits are 11, accumulator 3 is used as an index register. In this form of word addressing, known as "index register addressing", the displacement is treated as a signed, two's complement number which is added to the contents of the selected index register to produce a memory address. In index register addressing, the addition of the displacement to the contents of index register does not change the value contained in the index register.

The result of the addition performed in relative addressing and index register addressing is "clipped" to 15 bits. In other words, the high-order bit of the result is set to 0. For example, if accumulator 2 is to be used as an index register and contains the number 077774_8, and the displacement bits contain the number 012_8, then the result of the addition would be 000006_8, not 100006_8.

After one of the three types of addresses has been computed from the index and displacement bits, the indirect bit is tested. If this bit is zero, the address already computed is taken as the effective address. If the indirect bit is one, the word addressed by the result of the index and displacement bits is assumed to contain an address. In this word bit 0 is the indirect bit and bits 1-15 contain an address. If bit 0 of the referenced word is 1, another level of indirection is indicated, and bits 1-15 contain the address of the next word in the indirection chain. The processor will continue to follow this chain of indirect addresses until a word is retrieved with bit 0 set to 0. Bits 1-15 of this word are taken to be the effective address.

If an indirect address points to a location in the range $20\text{-}27_8$ (auto-increment locations); that word is fetched, the contents of the word are incremented by one and written back into the location. This updated value is then used to continue the addressing chain. If an indirect address points to a location in the range $30\text{-}37_8$ (auto-decrement locations), that word is fetched, the contents of the word are decremented by one and written back into the location. The updated value is then used to continue the addressing chain.

> **NOTE** When referencing auto-increment and auto-decrement locations, the state of bit 0 before the increment or decrement is the condition upon which the continuation of the indirection chain is based. For example: if an auto-increment location contains 177777_8, and the location is referenced as part of an indirection chain, location 0 will be the next address in the chain.



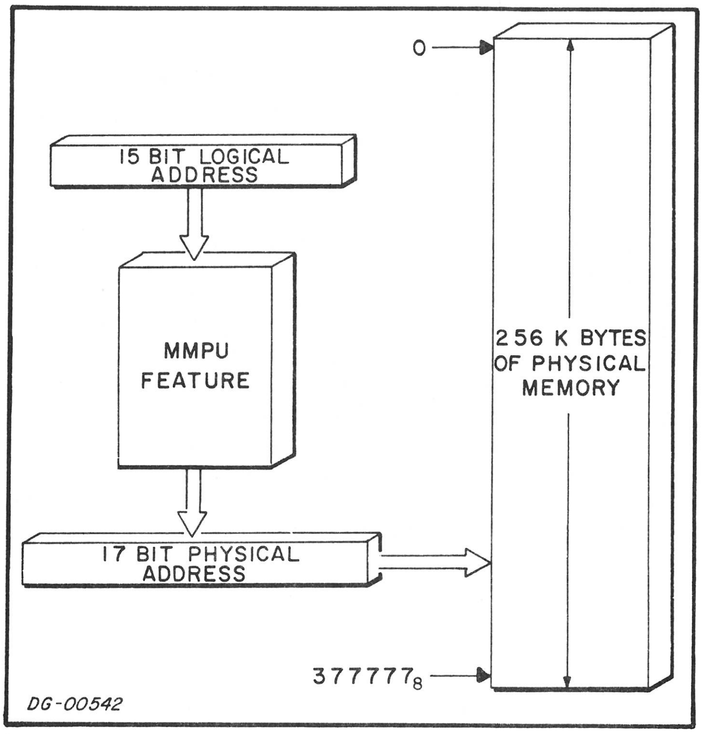
DG-00539

An effective address is always 15 bits in length. This means that an instruction which uses the effective address calculation can address any one of $32,768_{10}$ words. This gives rise to the concept of an "address space", which, in the NOVA line, contains 64K bytes or 32,768 2-byte words.

Addressing With The MMPU Feature

The concept of an address space was introduced in the discussion of effective address calculation. The "program" or "logical" address space is that amount of memory that can be referenced by instructions in a program. The "physical" address space is that amount of physical memory that can be referenced by the CPU. If the MMPU feature is not installed, the physical address space available to the CPU is 64K bytes or 32K words, and the logical address space is equal to the physical address space. Obviously, if the system contains less than 64K of physical memory, the usable address space is reduced, but the maximum physical address space of the NOVA 840 without the MMPU feature is 64K. With the MMPU feature installed, the logical address space is still 64K, but the maximum physical address space is increased to 256K bytes.

Installation of the MMPU feature has no effect on logical addressing. The addressing calculations remain the same. The MMPU feature comes into play when the CPU tries to use a 15-bit address to reference memory. The MMPU feature intercepts the memory reference and the 15-bit address. The MMPU feature then translates the 15-bit address into a 17-bit address with the aid of address translation hardware and the logical-to-physical address translation functions that have been set up by the supervisor program. The resultant 17-bit address is used to reference memory.



DG-00542

PROGRAM EXECUTION

Programs for the NOVA line consist of sequences of instructions that reside in main memory. The order in which these instructions are executed depends on a 15-bit counter called the "program counter". The program counter always contains the address of the instruction currently being executed. After the completion of each instruction the program counter is incremented by one and the next instruction is fetched from this address. This method of operation is called "sequential operation" and the instruction fetched from the location addressed by the incremented program counter is called the "next sequential instruction".

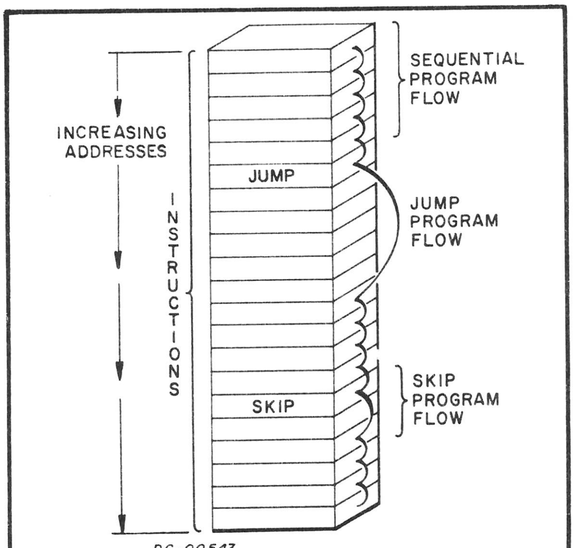
Program Flow Alteration

Sequential operation can be explicitly altered by the programmer in two ways. Jump instructions alter program flow by inserting a new value into the program counter. Conditional skip instructions can alter program flow by incrementing the program counter an extra time if a specified test condition is true. In the case of a conditional skip instruction when the test condition is true, the next sequential instruction is not executed because it is not addressed. After either a jump instruction or a successful conditional skip instruction, sequential operation continues with the instruction addressed by the updated value of the program counter.

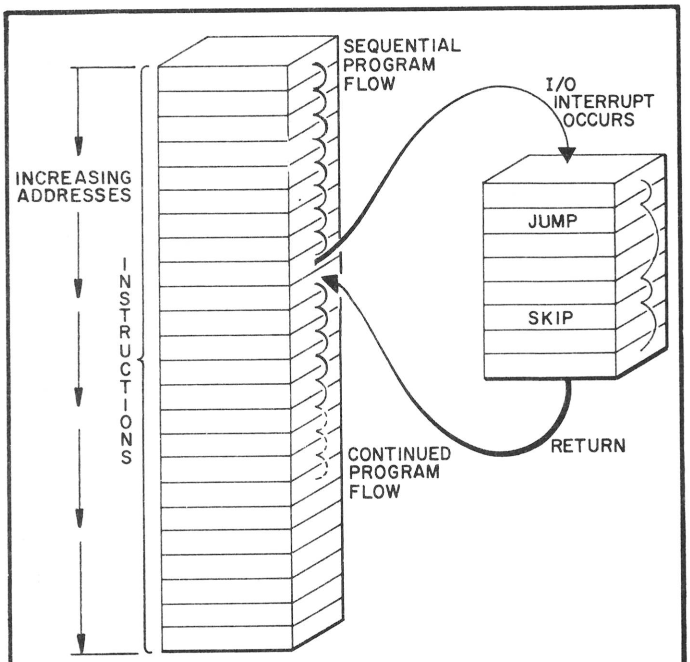
Bacause the program counter is 15 bits in length, it can address 32,768 separate memory locations. The next memory location after 77777_8 is location 0, and the location before 0 is location 77777_8. If the program counter rolls from 77777_8 to 0 in the course of sequential operation, no indication is given and processing continues with the location addressed by the updated value of the program counter.

Program Flow Interruption

The normal flow of a program may be interrupted by external or exceptional conditions such as I/O interrupts or various kinds of faults. In this case, the address of the next sequential instruction in the interrupted program is saved by the CPU so that the I/O handler or the various fault handlers can return control to the program at the correct point. Once the address of the next sequential instruction in the program has been placed in the program counter by the fault handler, sequential operation of the program resumes.



DG-00543



DG-00544

This page intentionally left blank.

SECTION 3

INSTRUCTION SETS

INTRODUCTION

The instruction set implemented on the NOVA line is divided into 5 instruction sets. There are instruction sets available for fixed point arithmetic, logical operations, program flow alteration, floating point arithmetic, and I/O operations. In addition, instruction sets which are a mixture of I/O instructions are available for programming the MMPU and MAP features, the RTC feature, the power fail/auto-restart feature, and certain CPU functions.

INSTRUCTION FORMATS

There are four different formats for instructions on the NOVA line. These formats allow an extensive instruction set while still keeping the instruction length to one word. The four formats and their general layouts are described below.

NO ACCUMULATOR-EFFECTIVE ADDRESS

0	0	0	OP CODE		@	INDEX		DISPLACEMENT							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

In the No Accumulator-Effective Address format instructions, bits 0-2 are 000, and bits 3-4 contain the operation code. The effective address is computed from bits 5-15 as described under "Effective Address Calculation".

ONE ACCUMULATOR-EFFECTIVE ADDRESS

0	OP CODE		AC		@	INDEX		DISPLACEMENT							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

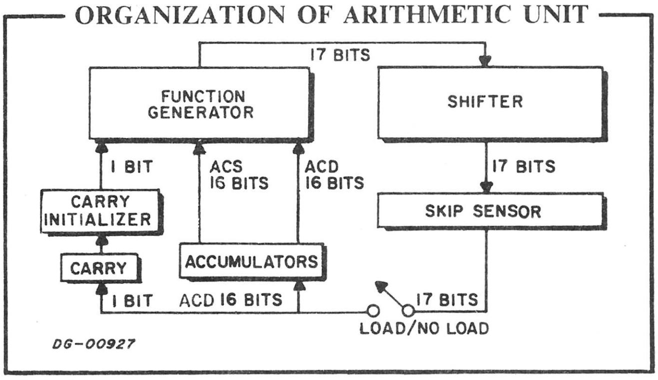
In the One Accumulator-Effective Address format instructions, bit 0 is 0, and bits 1-2 contain the operation code. Bits 3-4 specify the accumulator for the operation. The effective address is computed from bits 5-15 as described under "Effective Address Calculation".

TWO ACCUMULATOR-MULTIPLE OPERATION

1	ACS		ACD		OP CODE			SH		C		#	SKIP		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

In the Two Accumulator-Multiple Operation format instructions, bit 0 is 1, bits 1 and 2 specify the source accumulator, bits 3 and 4 specify the destination accumulator, bits 5-7 contain the operation code, bits 8 and 9 specify the action of the shifter, bits 10 and 11 specify the value to which the carry bit will be initialized, bit 12 specifies whether or not the result will be loaded into the destination accumulator, and bits 13-15 specify the skip test. Each instruction in this format utilizes an arithmetic unit whose logical organization is illustrated below.

ORGANIZATION OF ARITHMETIC UNIT



Each instruction specifies two accumulators to supply operands to the function generator, which performs the function specified by bits 5-7 of the instruction. The function generator also produces a carry bit whose value depends upon three quantities: an initial value specified by the instruction, the function performed, and the result obtained. The initial value may be derived from the previous value of the carry bit, or the instruction may specify an independent value.

The 17-bit output of the function generator, made up of the carry bit and the 16-bit function result, then goes to the shifter. In the shifter, the 17-bit result can be rotated one place right or left, or the

two 8-bit halves of the function result can be swapped without affecting the carry bit. The 17-bit output of the shifter can then be tested for a skip. The skip sensor can test whether the carry bit or the rest of the 17-bit result is or is not equal to zero. After the skip sensor has tested the shifter output, it can be loaded into the carry bit and the destination accumulator. Note, however, that loading is not necessary. An instruction in this format can perform a complicated arithmetic and shifting operation and test the result for a skip without affecting the carry bit or either of the operands.

INPUT/OUTPUT

0	1	1	AC		OP CODE			CONTROL		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

In the Input/Output format instructions, bits 0-2 are 011, bits 3-4 specify the accumulator for the operation, bits 5-7 contain the operation code, bits 8-9 specify the control signal to be used, and bits 10-15 contain the device code of the referenced device.

CODING AIDS

In the descriptions of the separate instructions, the general form of how the instruction is coded in assembly language is given along with the instruction. The general form of how an instruction may be coded has the following format:

MNEMONIC<optional mnemonics> OPERAND STRING

The mnemonic must be coded exactly as shown in the instruction description. Some instructions have optional mnemonics that may be appended to the main mnemonic if the option is desired. The operand string is made up of the operands for the given instruction.

The symbols <> and = are used in this manual to aid in defining the instructions. These symbols are not coded; they act only to indicate how an assembly language instruction may be written. Their general definition is given below.

<> Indicates optional operands or mnemonics. The operand enclosed in the brackets (e.g., <#>) may be coded or not, depending on whether or not the associated option is desired.

= Indicates specific substitution is required. Substitute the desired accumulator, address, name, number, or mnemonic.

The following abbreviations are used throughout this manual:

AC	=	Accumulator
ACS	=	Source Accumulator
ACD	=	Destination Accumulator
FPAC	=	Floating Point Accumulator

In the instructions that utilize an effective address, the following coding conventions are used:

The indirect bit (bit 5) is set to 1 by coding the symbol @ anywhere in the effective address operand string.

The index bits are set by coding a comma followed by one of the digits 0-3 as the last operand of the operand string. If no index is coded, the bits are set to 00. The character "period" (.) can be used to set the index bits to 01. "Period" can be read to mean "address of the current instructions". When the period is used, it is followed by either a plus or a minus sign followed by the displacement e.g., ".+7", or ".-2".

The displacement is coded as a signed number in the current assembler radix. This radix is the numbering system in which the programmer supplies numbers to the assembler. The default radix is Base 8 or octal. The assembler radix can be changed by using the RADIX statement.

The assembler available with the NOVA line allows the programmer to place labels on instructions or locations in memory. When the assembler comes upon a label in the operand string of an effective address instruction, it automatically sets the index and displacement bits to the correct values. For a detailed discussion of the features and operation of the NOVA line assembler, see the assembler manual (DGC 093-000017).

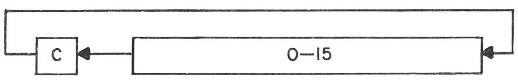
The fixed point and logical instructions which use the two accumulator-multiple operation format have several options that can be obtained by appending suffixes to the instruction mnemonic and by coding optional operands in the operand string. The characters to be coded are given below with their results.

The characters in the column titled "class abbreviation" refer to specific fields in the two accumulator-multiple operation format. The characters in the column titled "coded character" show the various characters which may be coded for this option. The numbers in the column titled "result bits" show the bit settings in these fields resulting from each coded character. The comments in the column titled "operation" describe the effect of these bit settings.

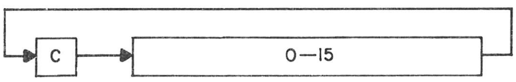
CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
C	(option omitted)	00	Do not initialize the carry bit.
	Z	01	Initialize the carry bit to 0.
	O	10	Initialize the carry bit to 1.
	C	11	Initialize the carry bit to the complement of its present value.
SH	(option omitted)	00	Leave the result of the arithmetic or logical operation unaffected.
	L	01	Combine the carry and the 16-bit result into a 17-bit number and rotate it one bit left.
	R	10	Combine the carry and the 16-bit result into a 17-bit number and rotate it one bit right.
	S	11	Exchange the two 8-bit halves of the 16-bit result without affecting the carry.
#	(option omitted)	0	Load the result of the shift operation into ACD.
	#	1	Do not load the result of the shift operation into ACD.

The following diagrams illustrate the operation of the shifter.

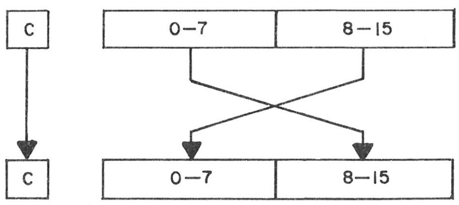
Coded Character	Shifter Operation
L	Left rotate one place. Bit 0 is rotated into the carry position, the carry bit into bit 15.



Coded Character	Shifter Operation
R	Right rotate one place. Bit 15 is rotated into the carry position, the carry bit into bit 0.



Coded Character	Shifter Operation
S	Swap the halves of the 16-bit result. The carry is not affected.



The following operands initiate operations that test the result of the shift operation. If the tested condition is true, the next sequential instruction is skipped.

CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
SKIP	(option omitted)	000	Never skip.
	SKP	001	Always skip.
	SZC	010	Skip if carry $= 0$.
	SNC	011	Skip if carry $\neq 0$.
	SZR	100	Skip if result $= 0$.
	SNR	101	Skip if result $\neq 0$.
	SEZ	110	Skip if either carry or result $= 0$.
	SBN	111	Skip if both carry and result $\neq 0$.

As an example of how to use these tables, assume that accumulator 3 contains a signed, two's complement number. Now consider the problem of determining whether this number is positive or negative. One way to determine this would be to place the number zero in another accumulator and use the SKIP IF ACS GREATER THAN ACD instruction, but this requires an extra instruction and also destroys the previous contents of the other accumulator. Another way to determine the sign of the number in accumulator 3 is to use the MOVE instruction and the power of the two accumulator-multiple operation format. With the MOVE instruction, the contents of AC3 can be placed in the shifter and shifted one bit to the left. This places the sign bit in the carry bit. The carry bit can then be tested for zero. In order to preserve the number in AC3, the instruction can prevent the output of the shifter from being loaded back into AC3.

The general form of the MOVE instruction is:

MOV<c><sh><#> acs,acd<,skip>

The general bit pattern of the MOVE instruction is:

1	ACS		ACD		0	1	0	SH		C		#	SKP		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

To shift the number in AC3 one bit left without destroying the number, and skip the next sequential instruction if the bit shifted into the carry bit is zero, the following instruction could be coded:

MOVL# 3,3,SZC

This instruction would assemble into the following bit pattern:

1	1	1	1	1	0	1	0	0	1	0	0	1	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

FIXED POINT ARITHMETIC

The fixed point instruction set performs binary arithmetic on operands in accumulators. The operands are 16 bits in length and can be either signed or unsigned. The instruction set provides for loading, storing, adding, and subtracting.

LOAD ACCUMULATOR

LDA ac,<@>displacement<,index>

0	0 1	AC	@	INDEX	DISPLACEMENT
0	1 2	3 4	5	6 7	8 9 10 11 12 13 14 15

The word addressed by the effective address, "E", is placed in the specified accumulator. The previous contents of the AC are lost. The contents of the location addressed by "E" remain unchanged.

STORE ACCUMULATOR

STA ac,<@>displacement<,index>

0	1 0	AC	@	INDEX	DISPLACEMENT
0	1 2	3 4	5	6 7	8 9 10 11 12 13 14 15

The contents of the specified accumulator are placed in the word addressed by the effective address, "E". The previous contents of the location addressed by "E" are lost. The contents of the specified accumulator remain unchanged.

ADD

ADD<c><sh><#> acs,acd<,skip>

1	ACS	ACD	1 1 0	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to the specified value. The number in ACS is added to the number in ACD and the result is placed in the shifter. If the addition produces a carry of 1 out of the high-order bit, the carry bit is complemented. The specified shift operation is performed and the result of the shift is placed in ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

NOTE If the sum of the two numbers being added is greater than $65,535_{10}$, the carry bit is complemented.

SUBTRACT

SUB<c><sh><#> acs,acd<,skip>

1	ACS	ACD	1 0 1	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to its specified value. The number in ACS is subtracted from the number in ACD by taking the two's complement of the number in ACS and adding it to the number in ACD. The result of the addition is placed in the shifter. If the operation produces a carry of 1 out of the high-order bit, the carry bit is complemented. The specified shift operation is performed and the result of the shift is placed in ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

NOTE If the number in ACS is less than or equal to the number in ACD the carry bit is complemented.

NEGATE

NEG<c><sh><#> acs,acd<,skip>

1	ACS	ACD	0 0 1	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to the specified value. The two's complement of the number in ACS is placed in the shifter. If the negate operation produces a carry of 1 out of the high-order bit, the carry bit is complemented. The specified shift operation is performed and the result is placed in ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

NOTE If ACS contains 0, the carry bit is complemented.

ADD COMPLEMENT

ADC<c><sh><#> acs,acd<,skip>

1	ACS	ACD	1 0 0	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to the specified value. The logical complement of the number in ACS is added to the number in ACD and the result is placed in the shifter. If the addition produces a carry of 1 out of the high-order bit, the carry bit is complemented. The specified shift operation is performed, and the result of the shift is loaded into ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

NOTE If the number in ACS is less than the number in ACD, the carry bit is complemented.

MOVE

MOV<c><sh><#>   acs, acd<, skip>

1	ACS		ACD		0	1	0	SH		C		#	SKP		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The carry bit is initialized to the specified value. The contents of ACS are placed in the shifter. The specified shift operation is performed and the result of the shift is loaded into ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

Example:

The MOVE instruction can be used to perform a signed divide by a power of 2 without using another accumulator. The following sequence of instructions will divide the signed, two's complement number in AC2 by 4 without using another accumulator.

```
MOVL#  2,2,SZC ;SKIP IF POSITIVE
MOVOR  2,2,SKP ;SHIFT RIGHT WITH 1 AND
               ; SKIP
MOVZR  2,2,SKP ;SHIFT RIGHT WITH 0 AND
               ; SKIP
MOVOR  2,2,SKP ;SHIFT RIGHT WITH 1 AND
               ; SKIP
MOVZR  2,2     ;SHIFT RIGHT WITH 0 AND
               ; DON'T SKIP
```

Shifting a number right one bit position is equivalent to dividing the number by 2 and rounding down. To perform division of a signed number in this manner, the bit shifted into the high-order bit must be equal to the sign bit. The first instruction determines whether to shift in a 0 or a 1.

INCREMENT

INC<c><sh><#>   acs, acd<, skip>

1	ACS		ACD		0	1	1	SH		C		#	SKP		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The carry bit is initialized to the specified value. The number in ACS is incremented by one and the result is placed in the shifter. If the incrementation produces a carry of 1 out of the high-order bit, the carry is complemented. The specified shift operation is performed, and the result of the shift is loaded into ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

NOTE If the number in ACS is 177777_8 the carry bit is complemented.

LOGICAL OPERATIONS

The logical instruction set performs logical operations on operands in accumulators. The operands are 16 bits long and are treated as unstructured binary quantities. The logical operations included in this set are: AND, and COMPLEMENT.

COMPLEMENT

COM<c><sh><#> acs,acd<,skip>

1	ACS	ACD	0 0 0	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to the specified value. The logical complement of the number in ACS is placed in the shifter. The specified shift operation is performed and the result is placed in ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

AND

AND<c><sh><#> acs,acd<,skip>

1	ACS	ACD	1 1 1	SH	C	#	SKP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

The carry bit is initialized to the specified value. The logical AND of ACS and ACD is placed in the shifter. Each bit placed in the shifter is 1 only if the corresponding bit in both ACS and ACD is one; otherwise the result bit is 0. The specified shift operation is performed and the result is placed in ACD if the no-load bit is 0. If the skip condition is true, the next sequential instruction is skipped.

PROGRAM FLOW ALTERATION

As stated previously, the normal method of program execution is sequential. That is, the processor will continue to retrieve instructions from sequentially addressed locations in memory until directed to do otherwise. Instructions are provided in the instruction set that alter this sequential flow. Program flow alteration is accomplished by placing a new value in the program counter. Sequential operations will then continue with the instruction addressed by this new value. Instructions are provided that change the value of the program counter, change the value of the program counter and save a return address, or modify a memory location by incrementing or decrementing and skip the next sequential instruction if the result is zero.

JUMP

JMP <@ >displacement<, index>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	@	INDEX		DISPLACEMENT							

The effective address, "E" is computed and placed in the program counter. Sequential operation continues with the word addressed by the updated value of the program counter.

JUMP TO SUBROUTINE

JSR <@ >displacement<, index>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	@	INDEX		DISPLACEMENT							

The effective address, "E" is computed. Then the present value of the program counter is incremented by one and the result is placed in AC3. "E" is then placed in the program counter and sequential operation continues with the word addressed by the updated value of the program counter.

NOTE The computation of "E" is completed before the incremented program counter is placed in AC3.

INCREMENT AND SKIP IF ZERO

ISZ <@ >displacement<, index>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	0	@	INDEX		DISPLACEMENT							

The word addressed by "E" is incremented by one and the result is written back into that location. If the updated value of the location is zero, the next sequential instruction is skipped.

DECREMENT AND SKIP IF ZERO

DSZ <@ >displacement<, index>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	@	INDEX		DISPLACEMENT							

The word addressed by "E" is decremented by one and the result is written back into that location. If the updated value of the location is zero, the next sequential instruction is skipped.

SECTION 4

INPUT/OUTPUT

INTRODUCTION

In order for the processor to perform useful work for the user, there must be some method for the program to transfer information outside the machine. The Input/Output (I/O) instruction set provides this facility. There are eight I/O instructions which allow the program to communicate with I/O devices, control the I/O interrupt system, control certain processor options, and to perform certain processor functions.

The NOVA line has a 6-bit device selection network, corresponding to bits 10-15 in the I/O instruction format. Each device is connected to this network in such a way that each device will only respond to commands with its own device code. Each device also has two flags, Busy and Done, which control its operation. When Busy and Done are both 0, the device is idle and cannot perform any operations. To start a device, the program must set Busy to 1 and set Done to 0. When a device has finished its operation, it sets Busy to 0 and Done to 1. The case of Busy and Done both set to 1 is a meaningless situation and will produce unpredictable results.

The format for the I/O instructions is illustrated below.

0	1	1	AC		OP CODE			CONTROL		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 0-2 are 011, bits 3-4 specify the AC, bits 5-7 contain the operation code, bits 8-9 control the Busy and Done flags in the device, and bits 10-15 specify the code of the device. The six bits provided for the device code in the I/O format mean that 64 unique device codes are available for use. Some of these device codes, however, are reserved for the CPU and certain processor options. The remaining device codes are available for referencing I/O units. Some of the codes have been assigned to specific devices by Data General and the assembler recognizes mnemonics for these devices. A complete listing of device codes, the devices assigned to these codes, and the mnemonics assigned to the devices is available in Appendix A.

OPERATION OF I/O DEVICES

In general, the operation of all I/O devices is done by manipulation of the Busy and Done flags. In order to operate a device, the program must first ensure that the device is not currently performing some operation. After the program has determined that the device is available, it can start an operation on the device by setting Busy to 1 and Done to 0. Once a device has completed its operation, and set Busy to 0 and Done to 1, it is available for another operation. The program can determine this condition in one of two ways. By using the I/O SKIP instruction, the program can test the status of the Busy and Done flags. Another way is to utilize the interrupt system that is standard on the NOVA line of computers. The interrupt system is made up of an interrupt request line to which each I/O device is connected, an Interrupt On flag in the CPU, and a 16-bit interrupt priority mask. The Interrupt On flag controls the status of the interrupt system. If the flag is set to 1, the CPU will respond to and process interrupts. If the flag is set to 0, the CPU will not respond to any interrupts. An interrupt is initiated by an I/O device when it completes its operation. Upon completing the operation, the device sets Busy to 0 and Done to 1. At this time, the device also places an interrupt request on the interrupt request line, provided that the bit in the interrupt priority mask which corresponds to the priority level of the device is 0. If the mask bit is 1, the device sets Busy to 0 and Done to 1, but does not place an interrupt request on the interrupt request line.

If the Interrupt On flag is 1 at the time the processor completes execution of any instruction, the processor honors any request on the interrupt request line. If the Interrupt On flag is 0, the CPU does not look at the interrupt request line; it just goes on to the next sequential instruction. The CPU honors an interrupt request by setting the Interrupt On flag to 0 so that no interrupts can interrupt the first part of the interrupt service routine. The CPU then places the updated program counter into memory location 0 and executes a "JMP@1" instruction. It is assumed that location 1 contains the address, either direct or indirect, of the interrupt service routine.

Once the CPU has transferred control to the interrupt service routine, it is up to that routine to save any accumulators that will be used, save the carry bit if it will be used, determine which device requested the interrupt, and then service the interrupt. The determination of which device needs service can be done by I/O SKIP instructions or the routine can use the INTERRUPT ACKNOWLEDGE instruction.

The INTERRUPT ACKNOWLEDGE instruction returns the 6-bit device code of the device requesting the interrupt. If more than one device is requesting service, the code returned is the code of that device requesting an interrupt which is physically closest to the CPU on the I/O bus. After servicing the device, the interrupt routine should restore all saved values, set the Interrupt On flag to 1, and return to the interrupted program. The instruction that sets the Interrupt On flag to 1 (INTERRUPT ENABLE) allows the processor to execute one more instruction before the next interrupt can take place. In order to prevent the interrupt service routine from going into a loop, this next instruction should be the instruction that returns control to the interrupted program. Since the updated value of the program counter was placed in location 0 by the CPU upon honoring the interrupt, all the interrupt routine has to do, after restoring the AC's and the carry bit, is execute an INTERRUPT ENABLE instruction, a "JMP@0" instruction and control will be returned to the interrupted program.

PRIORITY INTERRUPTS

If the Interrupt On flag remains 0 through the interrupt service routine, the interrupt routine cannot be interrupted and there is only one level of device priority. This level is determined by either the order in which the I/O SKIP instructions are issued or (if INTERRUPT ACKNOWLEDGE is used) by the physical location of the devices on the bus. In a system with devices of widely differing speed, such as a teletypewriter versus a fixed head disc, the programmer may wish to set up a multiple level interrupt scheme. Hardware and instructions are available that allow the implementation of sixteen levels of priority interrupts.

Each of the I/O devices is connected to a bit in the 16-bit priority mask. Devices which operate at roughly the same speed are connected to the same bit in the mask. Even though the standard mask bit assignments have the higher numbered bits assigned to lower speed devices, no implicit priority ordering is intended. The manner in which these priority levels are ordered is completely up to the programmer. The listing of device codes in Appendix A also contains the standard Data General mask bit assignments.

The condition of the priority mask is altered by the MASK OUT instruction. If a bit in the priority mask is set to 1, then all devices in the priority level corresponding to that bit will be prevented from requesting an interrupt when they complete an operation. In addition, all pending interrupt requests from devices in that priority level are disabled.

To implement a multiple priority level interrupt handler, the interrupt handler must be written in such a way that it may be interrupted without damage. For this to be possible, the main interrupt routine must save the state of the machine upon receiving control. The state of the machine consists of the four accumulators, the carry bit, and the return address. This information should be stored in a unique place each time the interrupt handler is entered so that one level of interrupt does not overlay the return information that belongs to a lower priority level. After saving the return information, the interrupt routine must determine which device requires service and jump to the correct service routine. This can be done in the same manner as for a single level interrupt handler.

After the correct service routine has received control, that routine should save the current priority mask, establish the new priority mask, and enable the interrupt system with the INTERRUPT ENABLE instruction. After servicing the interrupt, the routine should disable the interrupt system with the INTERRUPT DISABLE instruction, reset the priority mask, restore the state of the machine, enable the interrupt system, and return control to the interrupted program.

DATA CHANNEL

Handling data transfers between external devices and memory under program control requires an interrupt plus the execution of several instructions for each word transferred. To allow greater transfer rates the NOVA line contains a data channel through which a device, at its own request, can gain direct access to memory using a minimum of processor time.

When a device is ready to send or receive data, it requests access to memory via the channel. At the beginning of every memory cycle the processor synchronizes any requests that are then being made. At certain specified points during the execution of an instruction, the CPU pauses to honor all previously synchronized requests. When a request is honored, a word is transferred directly via the channel from the device to memory or from memory to the device without specific action by the program. All requests are honored according to the relative position of the requesting devices on the I/O bus. That device requesting data channel service which is physically closest on the bus in serviced first, then the next closest device, and so on, until all requests have been honored. The synchronization of new requests occurs concurrently with the honoring of other requests, so if a device continually requests the data channel, that device can prevent all devices further out on the bus from gaining access to the channel.

Following completion of an instruction, the processor handles all data channel requests, and then honors all outstanding I/O interrupt requests. After all data channel and I/O interrupt requests have been serviced, the processor continues with the next sequential instruction. The data channel is fully described in the "Programmer's Reference Manual for Peripherals", ordering number 015-000021.

CODING AIDS

The set of I/O instructions has options that can be obtained by appending mnemonics to the standard mnemonic. These optional mnemonics and their result are given below.

CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
f	(option omitted)	00	Does not affect the Busy and Done flags.
	S	01	Start the device by setting Busy to 1 and Done to 0.
	C	10	Idle the device by setting both Busy and Done to 0.
	P	11	Pulse the special in-out bus control line. The effect, if any, depends upon the device.

I/O INSTRUCTIONS

DATA IN A

DIA<f> ac, device

0	1	1	AC	0	0	1	F	DEVICE CODE
0	1	2	3 4	5	6	7	8 9	10 11 12 13 14 15

The contents of the A input buffer in the specified device are placed in the specified AC. After the data transfer, the Busy and Done flags are set according to the function specified by F.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device. Bits in the AC that do not receive data are set to 0.

DATA IN B

DIB<f> ac, device

0	1	1	AC	0	1	1	F	DEVICE CODE
0	1	2	3 4	5	6	7	8 9	10 11 12 13 14 15

The contents of the B input buffer in the specified device are placed in the specified AC. After the data transfer, the Busy and Done flags are set according to the function specified by F.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device. Bits in the AC that do not receive data are set to 0.

DATA IN C

DIC<f> ac, device

0	1	1	AC		1	0	1	F		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the C input buffer in the specified device are placed in the specified AC. After the data transfer, the Busy and Done flags are set according to the function specified by F.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device. Bits in the AC that do not receive data are set to 0.

DATA OUT A

DOA<f> ac, device

0	1	1	AC		0	1	0	F		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the specified AC are placed in the A output buffer of the specified device. After the data transfer, the Busy and Done flags are set according to the function specified by F. The contents of the specified AC remain unchanged.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device.

DATA OUT B

DOB<f> ac, device

0	1	1	AC		1	0	0	F		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the specified AC are placed in the B output buffer of the specified device. After the data transfer, the Busy and Done flags are set according to the function specified by F. The contents of the specified AC remain unchanged.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device.

DATA OUT C

DOC<f> ac, device

0	1	1	AC		1	1	0	F		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The contents of the specified AC are placed in the C output buffer of the specified device. After the data transfer, the Busy and Done flags are set according to the function specified by F. The contents of the specified AC remain unchanged.

The number of data bits moved depends upon the size of the buffer and the mode of operation of the device.

The I/O SKIP instruction enables the programmer to make decisions based upon the values of the Busy and Done flags. Which test is performed is based upon the value of bits 8-9 in the instruction. Bits 8-9 can be set by appending an optional mnemonic to the I/O SKIP mnemonic. The optional mnemonics and their results are given below.

CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
t	BN	00	Tests for Busy = 1.
	BZ	01	Tests for Busy = 0.
	DN	10	Tests for Done = 1.
	DZ	11	Tests for Done = 0.

I/O SKIP

SKP<t> device

0	1	1	0	0	1	1	1	T		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If the test condition specified by T is true, the next sequential instruction is skipped.

NO I/O TRANSFER

NIO<f> device

0	1	1	0	0	0	0	0	F		DEVICE CODE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Busy and Done flags in the specified device are set according to the function specified by F.

CENTRAL PROCESSOR FUNCTIONS

I/O instructions with a device code of 77 perform a number of special functions rather than controlling a specific device. In all but the I/O SKIP instruction, I/O instructions with a device code of 77 use bits 8-9 to control the condition of the Interrupt On flag. An I/O SKIP instruction with a device code of 77 uses bits 8-9 to either test the state of the Interrupt On flag or to test the state of the Power Fail flag. The mnemonics are the same as for normal I/O instructions. The table below gives the result of these bits for instructions with a device code of 77.

CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
f	(omitted)	00	Does not affect the state of the Interrupt On flag.
	S	01	Set the Interrupt On flag to 1.
	C	10	Set the Interrupt On flag to 0.
	P	11	Does not affect the state of the Interrupt On flag.
t	BN	00	Tests for Interrupt On = 1.
	BZ	01	Tests for Power Fail = 1.
	DN	10	Tests for Power Fail ≠ 0.
	DZ	11	Tests for Power Fail = 0.

The device code of 77 deals mainly with processor functions and has, therefore, been given the mnemonic of CPU. In addition, many of the I/O instructions that reference this device code have been given special mnemonics. While these special mnemonics are functionally equivalent to the corresponding I/O instructions with a device code of 77, there is the following limitation; the mnemonics for controlling the state of the Interrupt On flag cannot be appended to them. If the programmer wishes to alter the state of the Interrupt On flag while performing a MASK OUT instruction, for example, he must issue the appropriate I/O instruction (DOB<f> ac, CPU) instead of the corresponding special mnemonic (MSKO ac, CPU). If the special mnemonic is used, bits 8-9 are set to 00. In describing the instructions, the special mnemonic for the corresponding I/O instruction will be given first, followed by the I/O instruction.

INTERRUPT ENABLE

INTEN

NIOS CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1

The Interrupt On flag is set to 1. If the state of the Interrupt On flag is changed by this instruction, the CPU allows one more instruction to execute before the first I/O interrupt can occur.

INTERRUPT DISABLE

INTDS

NIOC CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	0	1	1	1	1	1	1

The Interrupt On flag is set to 0.

READ SWITCHES

READS ac

DIA<f> ac, CPU

0	1	2	3–4	5	6	7	8–9	10	11	12	13	14	15
0	1	1	AC	0	0	1	F	1	1	1	1	1	1

The setting of the console data switches is placed in the specified AC. After the transfer, the Interrupt On flag is set according to the function specified by F.

INTERRUPT ACKNOWLEDGE

INTA ac

DIB<f> ac, CPU

0	1	2	3–4	5	6	7	8–9	10	11	12	13	14	15
0	1	1	AC	0	1	1	F	1	1	1	1	1	1

The six-bit device code of that device requesting an interrupt which is physically closest to the CPU on the bus is placed in bits 10-15 of the specified AC. Bits 0-9 of the specified AC are set to 0. After the transfer, the Interrupt On flag is set according to the function specified by F.

MASK OUT

MSKO ac

DOB<f> ac, CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	F		1	1	1	1	1	1

The contents of the specified AC are placed in the priority mask. After the transfer, the Interrupt On flag is set according to the function specified by F. The contents of the specified AC remain unchanged.

NOTE A 1 in any bit disables interrupt requests from devices in the corresponding priority level.

I/O RESET

IORST

DIC<f> ac, CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	1	F		1	1	1	1	1	1

The Busy and Done flags in all I/O devices are set to 0. The 16-bit priority mask is set to 0. The Interrupt On flag is set according to the function specified by F.

NOTES The assembler recognizes the instruction IORST as equivalent to DICC 0, CPU.

If the mnemonic DIC is used to perform this function, an accumulator must be coded to avoid assembly errors. Regardless of how the instruction is coded, during exexution, the AC field is ignored and the contents of the AC remain unchanged.

HALT

HALT

DOC<f> ac, CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	1	0	F		1	1	1	1	1	1

The Interrupt On flag is set according to the function specified by F and then the processor is stopped.

NOTE If the mnemonic DOC is used to perform this function, an accumulator must be coded to avoid assembly errors. During execution of this instruction, the AC field is ignored.

CPU SKIP

SKP<t> CPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	1	T		1	1	1	1	1	1

If the test condition specified by T is true, the next sequential instruction is skipped.

SECTION 5

PROCESSOR OPTIONS

INTRODUCTION

Optional equipment for the NOVA line computers includes a power monitor with the facility for automatic restart after a power failure, multiply/divide, real-time clock.

POWER FAIL

In the NOVA line, when power is turned off and then on again, core memory is unaltered. However, when the power is turned on, the state of the accumulators, the program counter, and the various flags in the CPU is indeterminate. The power fail option provides a "fail-soft" capability in the event of unexpected power loss.

In the event of power failure, there is a delay of one to two milliseconds before the processor shuts down. The power fail option senses the imminent loss of power, sets the Power Fail flag, and requests an interrupt. The interrupt service routine can then use this delay to store the contents of the accumulators, the carry bit, and the current priority mask. The interrupt service routine should also save location 0 (to enable return to the interrupted program), put a JUMP to the desired restart location in location 0, and then execute a HALT. One to two milliseconds is enough time to execute 200 to 1500 instructions depending on the processor, so there is more than enough time to perform the power fail routine.

When power is restored, the action taken by the automatic restart portion of the power fail option depends upon the position of the power switch on the front panel. If the switch is in the "on" position, the CPU remains stopped after power is restored. If the switch is in the "lock" position, then 50ms after power is restored, the CPU executes a "JMP 0" instruction, restarting the interrupted program.

The power fail option has no device code and no interrupt disable bit in the priority mask. It does not respond to the INTERRUPT ACKNOWLEDGE instruction. The Power Fail flag can be tested by the CPU SKIP instruction. Testing of the Power Fail flag is described below.

SKIP IF POWER FAIL FLAG IS ONE

SKPDN CPU

0	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If the Power Fail flag is 1 (i.e., power is failing), the next sequential instruction is skipped.

SKIP IF POWER FAIL FLAG IS ZERO

SKPDZ CPU

0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If the Power Fail flag is 0 (i.e., power is not failing), the next sequential instruction is skipped.

MULTIPLY/DIVIDE

Multiplication can be performed on the NOVA line by software routines that utilize the standard instruction set, but if many of these operations are required, a loss of efficiency can result. The multiply/divide option provides the capability of performing these operations in hardware, with a corresponding increase in CPU efficiency and utilization. Two versions of this option are available: one for the NOVA computer, and one for the rest of the computers in the NOVA line. The two versions of this option and the instructions for each are described below.

NOVA MULTIPLY/DIVIDE

The multiply/divide option for the NOVA computer is an I/O device and is controlled by I/O instructions. The device code for the NOVA computer multiply/divide option is 1. It has no Busy and Done flags and does not respond to the INTERRUPT ACKNOWLEDGE instruction. It has three buffers: A, B, and C that can be written and read using standard I/O instructions. Multiplication and division is controlled by the setting of the control field in the I/O instruction. The control field setting and the resulting operation are described below.

CLASS ABBREVIATION	CODED CHARACTER	RESULT BITS	OPERATION
f	(option omitted)	00	None
	S	01	The contents of the A and B buffers are treated as an unsigned, double length integer, with the A buffer being the left half and the B buffer being the right half. This number is divided by the unsigned integer contained in the C buffer. The quotient is placed in the B buffer and the remainder is placed in the A buffer. The contents of the C buffer remain unchanged.
	C	10	The A buffer is set to 0.
	P	11	The unsigned integers contained in the B and C buffers are multiplied together to form a double length, unsigned, intermediate result. The unsigned integer contained in the A buffer is added to this number and the final result is placed in the A and B buffers. The left half is placed in the A buffer and the right half is placed in the B buffer. The contents of the C buffer remain unchanged.

NON-NOVA MULTIPLY/DIVIDE

The multiply/divide option for the rest of the computers in the NOVA line is a part of the CPU. For compatibility, the instructions for the option are I/O instructions that reference device code 1. The assembler recognizes the mnemonics MUL and DIV for these operations. The Mnemonics and the I/O instructions generated along with a description of the instructions appear below.

MULTIPLY

MUL

DOCP 2, MDV

0	1	1	1	0	1	1	0	1	1	0	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The 16-bit unsigned number in AC1 is multiplied by the 16-bit unsigned number in AC2 to yield a 32-bit unsigned intermediate result. The 16-bit unsigned number in AC0 is added to the intermediate result to produce the final result. The final result is a 32-bit unsigned number and occupies AC0 and AC1. Bit 0 of AC0 is the high-order bit of the result and bit 15 of AC1 is the low-order bit. The contents of AC2 remain unchanged. The carry bit remains unchanged. Because the result is a double-length number, overflow cannot occur.

DIVIDE

DIV

DOCS 2, MDV

0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The 32-bit unsigned number contained in AC0 and AC1 is divided by the 16-bit unsigned number in AC2. The quotient and remainder are 16-bit unsigned numbers and are placed in AC1 and AC0, respectively. The carry bit is set to 0. The contents of AC2 remain unchanged.

NOTE Before the divide operation takes place, AC0 is compared to AC2. If the number in AC0 is greater than or equal to the number in AC2, an overflow condition is indicated. The carry bit is set to 1 and the operation is terminated. All operands remain unchanged.

REAL-TIME CLOCK

The Real-Time Clock (RTC) option available on the NOVA line generates a sequence of pulses that is independent of the CPU timing. It will generate I/O interrupts at any one of four program selectable frequencies. The Busy and Done flags of the RTC option are controlled by bits 8-9 of the I/O instruction. The RTC option is device code 14_8 and has the mnemonic RTC. The interrupt disable bit is priority mask bit 13.

Setting Busy allows the next pulse from the clock to set Done, and the RTC option requests an I/O interrupt if its interrupt disable bit is 0. A DATA OUT A instruction to select the clock frequency only has to be given once. After each interrupt, an NIOS instruction will set up the clock for the next interrupt.

When Busy is first set the first interrupt can come at any time up to the clock period. After the first interrupt has occurred, succeeding interrupts come at the clock frequency, provided that the program always sets Busy before the clock period expires. After power up or I/O reset, the clock is set to the line frequency. After power up the line frequency pulses are available immediately, but five seconds must elapse before a steady pulse train is available from the crystal for other frequencies.

The RTC frequency is selected by the following instruction.

SELECT RTC FREQUENCY

DOA<f> ac,RTC

0	1	2	3-4	5	6	7	8-9	10	11	12	13	14	15
1	0	0	AC	0	1	0	F	0	0	1	1	0	0

The clock frequency is set according to bits 14-15 of the specified AC. The contents of the specified AC remain unchanged.

AC BITS 14-15	FREQUENCY
00	AC line frequency
01	10Hz
10	100Hz
11	1000Hz

MEMORY MANAGEMENT AND PROTECTION UNIT

Background to Address Translation

The concept behind the Memory Management and Protection Unit (MMPU) available with the NOVA 830 and NOVA 840 computers is that of "Logical-to-Physical Address Translation". The amount of memory required by a user's program is defined to be his "logical address space". This space may be as large as 32 1K pages. The areas of physical storage assigned to the user are defined to be his "physical address space". The address translation function that converts addresses in the logical space to addresses in the physical space is called the "address map" for that user. Each user has his own, unique logical-to-physical address map. In addition, there is a map for the data channel which can be, but does not have to be equal to the user map. The multiprogramming operating system determines what these maps are to be, and then transmits this information to the MMPU. The following instruction shows a possible two-user configuration.

Figure 1 shows a 128K physical address space and its utilization by a two-user multiprogramming system. The supervisor resides in pages 0-7 of physical space. The first 16 pages of user #1 are in pages 8-23 of physical memory. The remaining 16 pages of the address space for user #1 reside in pages 40-55 of physical space. User #2 also has his 32K of logical space split into two areas. Pages 0-15 of user #2 are in pages 24-39 of physical space and pages 24-39 of physical space and pages 16-31 of user #2 are in pages 56-71 of physical space. The data channel is capable of servicing both users. Any data channel reference to pages 0-15 of logical space will be mapped to pages 0-15 of the logical space of user #1. Any data channel reference to logical pages 16-31 will be mapped to pages 0-15 of the logical space of user #2.

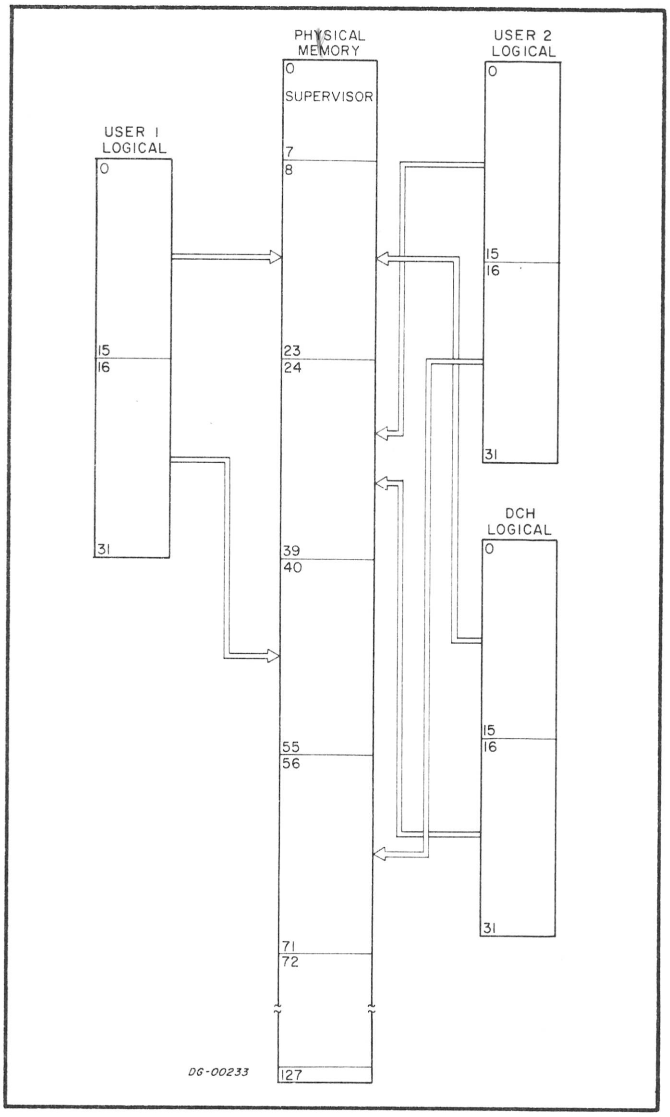


Figure 1 Logical-to-Physical Address Translation

In order to manage memory efficiently, the operating system makes use of the validity and write protect features of the MMPU. Figure 2 shows a two-user configuration where these features are used.

In Figure 2, a "W" in a page means the page is write-protected. By convention, mapping a logical page to physical page 127 and write protecting it makes that page validity protected. Both users have declared that page 1 of their logical space is to be write-protected.

Physical page 8 is the logical page 1 for user #1 and physical page 10 is the logical page 1 for user #2. User #1 is only using 13 pages of his 32 page logical address space, so logical pages 13-31 have been declared invalid for him. Any reference by user #1 to logical pages 13-31 will cause a validity error. User #2 is only using 21 pages of his logical address space, so logical pages 21-31 of his logical space have been declared invalid. Any reference by user #2 to logical pages 21-31 will result in a validity error.

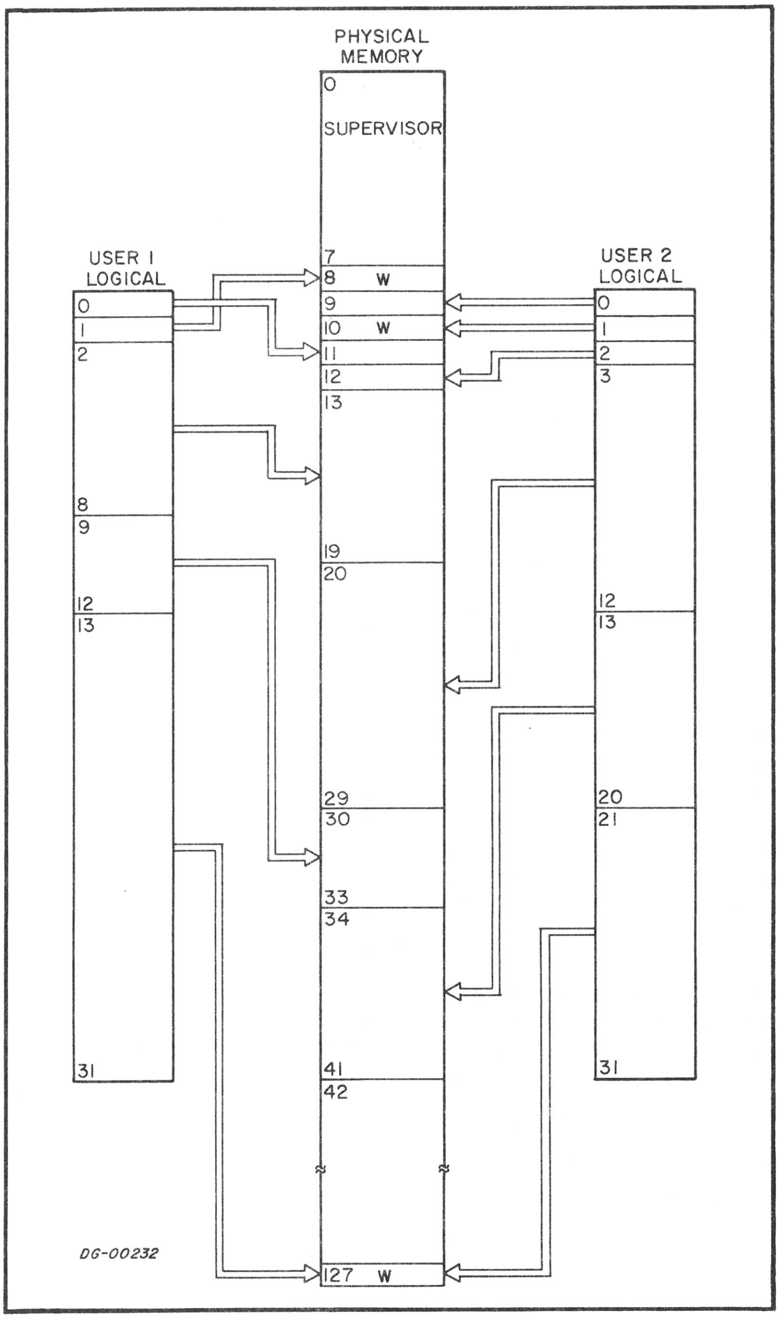


Figure 2 Logical-to-Physical Address Translation With Write and Validity Protection

The MMPU resides between the memory and the CPU, and the memory and the data channel, and is transparent to all of them. When either the CPU or the data channel requests a memory operation, the MMPU intercepts and services the request. the MMPU translates the 15 bit logical address coming from the CPU or the data channel into a 17 bit physical address. The memory operation is then performed using this 17 bit address. The memory access cycle time is unchanged.

The MMPU operates in two modes called user mode and supervisor mode. In user mode, all logical addresses coming from the CPU are translated using the user map. Checking is also performed for all protection features that are enabled. In supervisor mode, the user map for logical pages 0-30 is disabled and no protection checking is performed. All addresses in the range 76000_8-77777_8 will be translated using the user map for logical page 31. This enables the supervisor to access portions of user space while in supervisor mode, without resorting to lengthy use of the ENABLE SINGLE CYCLE instruction. The data channel map can be enabled or disabled in either of these modes.

The mapping information needed to service a CPU or data channel request is given to the MMPU by the operating system through I/O instructions that reference the MMPU. This information is transmitted before the supervisor enables either the user map or the data channel map.

When power is first turned on, or after an IORST instruction, the MMPU is in the supervisor mode and the data channel map is disabled. Logical page 31 is mapped to physical page 31. On power up, the user map, data channel map, and the device protect codes are undefined. After the first LOAD MAP instruction, logical page 31 is mapped according to whatever address is in that portion of the MMPU.

MMPU INSTRUCTION SET

The Memory Management and Protection Unit is programmed with ten I/O instructions. Through the use of these instructions, the multiprogramming operating system tells the MMPU what the address translation functions are to be. An address translation function is called a "map" and the two maps for the MMPU are the "user map" and the "data channel map". These two maps are separate and independent. They can be enabled concurrently. Enabling the user map allows the MMPU to translate addresses for the CPU. Enabling the data channel map allows the MMPU to translate addresses for the data channel.

The instructions for the MMPU are in the standard I/O format. The device code for the MMPU is 2.

LOAD MAP

DOA ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MMPU. The contents of the specified AC remain unchanged. The format of the AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	DCH	Logical Page					WP	Physical Page						

BITS	CONTENTS
0	Must be 0.
1	Must be 0.
2	0 = this instruction gives an address translation for the CPU (user map). 1 = this instruction gives an address translation for the data channel (data channel map).
3-7	Logical page number. This is an octal number in the range 0-37.
8	0 = no write-protect for this page. 1 = this page is to be write-protected. NOTE: A logical page is validity protected by mapping it to physical page number 127 and setting the write-protect bit. NOTE: If both the data channel bit and the write-protect bit are set, the write protect bit is ignored.
9-15	Physical page number. This is an octal number in the range 0-177.

This is the instruction that sets up the translation function from logical memory to physical memory. After this instruction is issued and the corresponding mapping feature enabled, any address in the 1K logical page is translated to the corresponding address in the 1K physical page.

Example:

Assume that a LOAD MAP instruction has been issued with bit 2=0, logical page=24, and physical page=105. With the user map enabled, the CPU requests data from location 50302. The MMPU will intercept this request, translate it, and retrieve the data from physical location 212302. This LOAD MAP instruction, mapping logical page 24 to physical page 105, would allow the mapping of all addresses in the range 50000-51777 of logical memory. Any request for an address in this 1K page would be translated to locations 212000-213777 in physical memory.

NOTE All numbers in the above example are octal.

LOAD DEVICE PROTECTION

DOA ac, MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MMPU. The contents of the specified AC remain unchanged. The format of the AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	Ignored			Device Class			Device Protect Bits							

BITS	CONTENTS
0	Must be 0.
1	Must be 1.
2-4	Ignored.
5-7	Device class. This is an octal number in the range 0-7. This is the most significant digit of the two-digit octal device code.
8-15	Device protect bits. The second digit of the two-digit octal device code is specified by the position in this field. A one in any bit protects the corresponding unit from receiving any commands directly from the user. For example, if bits 5-7 are 010 and bits 8-15 are 01010000, then devices 21 and 23 are protected. NOTE: Code 77 functions such as HALT, INTDS, IORST, etc., may be forbidden to the user by issuing this instruction with the contents of the specified AC set to 043401 (octal).

LOAD PROTECTION CONTROL

DOA ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MMPU. The contents of the specified AC remain unchanged. The format of the AC is as follows:

0	1	2	3	4	5	6–15
1	1	Defer	Write	I/O	DCH	Ignored

BITS	CONTENTS
0	Must be 1.
1	Must be 1.
2	0 = disable defer protection. 1 = enable defer protection.
3	Write-protect. 0 = disable write-protection. 1 = enable write-protection.
4	I/O protect. 0 = disable I/O protection. 1 = enable I/O protection.
5	Data channel map. 0 = disable data channel map. 1 = enable data channel map. NOTE: If this bit is 1, the data channel map is enabled immediately. NOTE: Each protection may be enabled independently of the others.
6-15	Ignored.

This instruction controls the data channel map and the protection features of the MMPU.

If a protection is disabled, the MMPU does no checking for it, and if a violation occurs, takes no action. If a protection is enabled, the MMPU checks each instruction for a violation of that protection and, if one is found, enters the supervisor mode and transfers control ("traps") to a specific physical location in the supervisor. These trap locations and the conditions that cause the trap are as follows:

TRAP LOCATION (octal)	TRAP CAUSE
40	I/O protect violation Validity violation
41	Runaway defer violation Write violation

These locations should contain jump instructions that will transfer control to supervisor routines that will determine the exact error and its severity and then take action.

The trap operation is equivalent to a direct jump to one of the trap locations.

ENABLE USER MAP

NIOS MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0

The address translation function for the CPU is enabled. Three fetch or defer cycles are allowed to elapse, then all CPU requests for memory are translated according to the previous LOAD MAP instructions. Entry into a user program should be done in the following manner:

```
                    ;SOME COMBINATION OF
                    ; LOAD PROTECTION
                    ; CONTROL, LOAD DE-
                    ; VICE PROTECTION, AND
                    ; LOAD MAP.
NIOS    2
INTEN
JMP     @ .+1
ADDR                ;USER START ADDRESS
```

The contents of ADDR and all succeeding CPU requests for memory are mapped.

INITIATE PAGE CHECK

DOA ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MMPU for later use by READ STATUS. The contents of the specified AC remain unchanged. The format of the AC is as follows:

0	1	2	3-7	8-15
1	0	DCH	Logical Page	Ignored

BITS	CONTENTS
0	Must be 1.
1	Must be 0.
2	Data channel bit. 0 = this instruction refers to the user map. 1 = this instruction refers to the data channel map.
3-7	Logical page. This is an octal number in the range 0-37, and is the number of the logical page for which status will be requested.
8-15	Ignored.

This instruction is used, in conjunction with the READ STATUS instruction, to determine the translation function for a logical page. The INITIATE PAGE CHECK instruction indicates to the MMPU which map and logical page should be referenced for the next READ STATUS instruction.

READ STATUS

DIC ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	1	0	1	0	0	0	0	0	0	1	0

The status bits for the MMPU and the write-protect bit and physical page number which correspond to the logical page number given in the last INITIATE PAGE CHECK instruction are placed in the specified AC. The previous contents of the specified AC are lost. The format of the data placed in the specified AC is as follows:

0	1	2	3	4	5	6	7	8	9-15
User	Write Vio	I/O Vio	Valid Vio	Single Cycle	Resvd	Defer	FP	WP	Physical Page

BITS	MEANING IF SET
0	User mode. The last program interrupt occurred while in user mode.
1	Write violation. A write violation has occurred.
2	I/O violation. An I/O violation has occurred.
3	Validity violation. A validity violation has occurred.
4	Single instruction map. The error occurred in the map cycle of an ENABLE SINGLE CYCLE instruction.
5	Reserved for future use.
6	Defer violation. The seventeenth level of a defer loop has been detected.
7	Floating point. A write-protect violation or validity violation occurred during a floating point unit data channel cycle.
8	Write-protect. This is the write-protect bit associated with this physical page.
9-15	Physical page. This is an octal number in the range 0-177 and is the number of the physical page which corresponds to the logical page given in the last INITIATE PAGE CHECK.

READ INSTRUCTION ADDRESS

DIA ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	0	1	0	0	0	0	0	0	1	0

The logical address of the instruction that caused the trap is placed in the specified AC. After the instruction, bit 0 of the specified AC is cleared and bits 1-15 contain the address as an octal number in the range 0-77777. The original contents of the specified AC are lost.

READ INVALID ADDRESS

DIB ac,MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	A	C	0	1	1	0	0	0	0	0	0	1	0

The logical address which caused the trap is placed in the specified AC. After the instruction, bit 0 of the specified AC is cleared and bits 1-15 contain the address as an octal number in the range 0-77777. The original contents of the specified AC are lost.

ENABLE SINGLE CYCLE

NIOP MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	1	0	0	0	0	1	0

The data fetch portion of an instruction is translated using the user map. Two fetch or defer cycles are allowed to elapse and the third fetch or defer cycle is translated using the user map. Succeeding fetch or defer cycles are mapped until an execute cycle occurs. After the first execute cycle, the user map is disabled and succeeding instructions are done in supervisor mode.

> **NOTE** No protection features are enabled during this mapping process.

This instruction can be used for at least two purposes:

a) to access data out of logical memory when not in user mode with a minimum of overhead.

b) to execute an instruction in the supervisor as if it were a user instruction.

> **NOTE** This instruction clears the status register.

Example:

The following instructions will load the contents of logical location 400_8 into AC0 while in supervisor mode:

```
NIOP   2
LDA    0,@.+2
JMP    .+2
000400
```

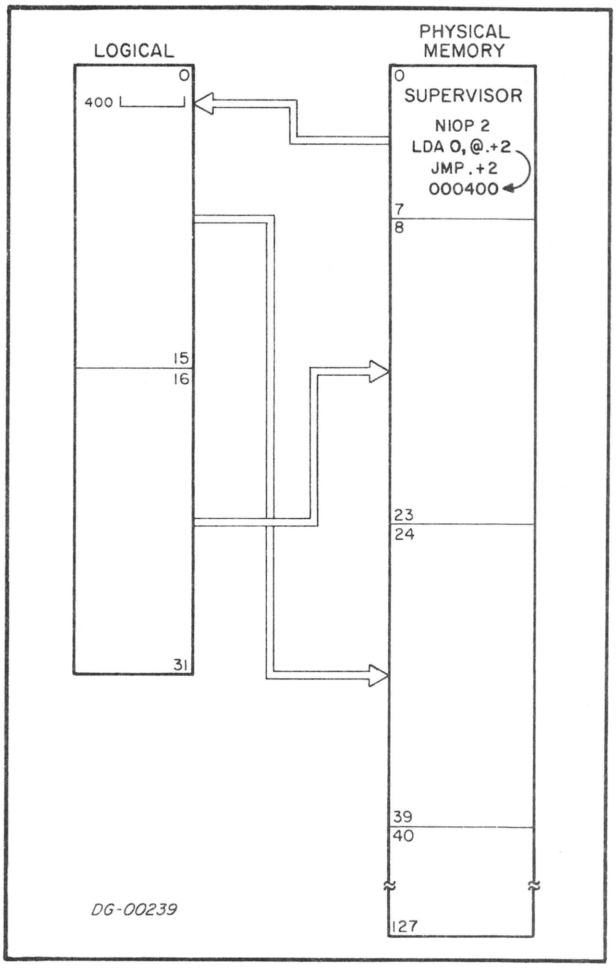


Figure 3 Graphic Representation of Example

SUPERVISOR CALL

NIOC MMPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0

The MMPU disables I/O requests, enters the supervisor mode and the next instruction is fetched from location 42 (octal) of physical memory. This instruction can be used to implement supervisor functions at the discretion of the individual installation.

SUPERVISOR PROGRAMMING FOR THE MEMORY MANAGEMENT AND PROTECTION UNIT

Setting Up For Translation

The information that allows the MMPU to translate addresses comes from the multiprogramming supervisor. The instructions used are LOAD MAP, LOAD DEVICE PROTECTION, and LOAD PROTECTION CONTROL. By using the LOAD MAP instruction, the supervisor gives the MMPU a beginning physical address for each of the 32 logical pages. At any single point in time, all 32 pages should be described. If there is no physical storage available to hold a logical page (for instance a machine with 16K of storage), then that page should be mapped to physical pagenumber 127 and write-protected. If this is the case, any attempted reference to this logical page will generate a validity trap. The LOAD MAP instruction is also used to direct the actions of the data channel. If the user is allowed to directly initiate data channel activity, the data channel map should be the same as the user map. If, however, the convention is that the supervisor will perform all I/O, the data channel map need not be the same as the user map.

LOAD DEVICE PROTECTION tells the MMPU what devices are to be declared inaccessible to the user. If the user tries to access a protected device and I/O protect is enabled, the MMPU will generate an I/O protect trap and the supervisor can take appropriate action. This allows the implementation of user dedicated devices.

> **NOTE** Although the 8020 Floating Point Processor is an I/O device and operates through the data channel, all floating point operations are processed using the user map.

After issuing the desired LOAD MAP and LOAD DEVICE PROTECTION instructions, the supervisor can direct which protect features are to be enabled by the LOAD PROTECTION CONTROL instruction. Each protect feature described in the LOAD PROTECTION CONTROL instruction can be enabled separately and independently of the others. When the supervisor has established the parameters for address translation, the ENABLE USER MAP instruction tells the MMPU to begin translating addresses. The MMPU will continue its mapping function until it senses a protection violation, at which point it will trap into the supervisor as described in the next section.

MMPU Protection Processing

In order to achieve efficient processing, the MMPU must perform its task until an exceptional condition arises and then tell the supervisor about the condition in a forthright manner. The MMPU does this through the use of two trap locations and three instructions. The trap locations are predetermined addresses in physical memory where the supervisor places instructions that are entries into supervisor routines. When the MMPU senses a violation of one of the enabled protect features, it will disable address translation, and direct the CPU to fetch the next instruction from one of these locations depending on the type of condition. The trap locations and their corresponding condition types are as follows:

PHYSICAL LOCATION (octal)	CONDITION
40	I/O protect or validity error
41	Runaway defer or write protect error

The MMPU instructions that allow the supervisor to determine what caused the trap are READ INSTRUCTION ADDRESS, READ INVALID ADDRESS, and READ STATUS. Upon entry into the I/O protect, validity error, runaway defer, or write-protect error routines, the supervisor can use these instructions to determine the type of error and its location. After learning this information, the supervisor can take appropriate action and restart or abort the user.

The MMPU performs checking only for these protection features that are enabled. The four types of protection and how they are handled in the MMPU are discussed below.

I/O Protection

If I/O protection is enabled, the MMPU decodes all I/O instructions and then looks in the I/O protect table to see if the referenced device is user protected. If it is not, the MMPU takes no action. If the device is protected, the MMPU does not allow execution of the instruction. Instead, the MMPU stores in both the INSTRUCTION ADDRESS and INVALID ADDRESS registers the logical address of the instruction, disables I/O interrupt request, enters the supervisor mode, and directs the CPU to fetch the next instruction from physical location 40 (octal).

Validity Protection

By convention, validity protection can not be disabled. Any logical page that is mapped to physical page 127 and write-protected, is assumed to be validity protected. The MMPU checks all CPU requests for invalid addresses. If the address is found to be valid, the MMPU proceeds with the required translation. If the address is invalid, the MMPU stores the invalid address in the INVALID ADDRESS register and stores the logical address of the instruction in the INSTRUCTION ADDRESS register. If the invalid address occurred in a defer or execute cycle, the instruction is allowed to complete with zeroes as data. Upon the completion of the instruction, the MMPU disables I/O interrupt requests, enters the supervisor mode, and directs the CPU to fetch the next instruction from physical location 40 (octal). If the invalid address occurred in a fetch cycle, the MMPU immediately disables the CPU interrupt system, enters the supervisor mode and directs the CPU to fetch the next instruction from physical location 40 (octal).

Runaway Defer Protection

If runaway defer protection is enabled, the MMPU checks memory references to see if they are part of a defer cycle. If the MMPU detects seventeen consecutive defer cycle memory requests, it traps. Upon receiving the seventeenth request, the MMPU stores the address of the instruction that started the defer loop in the INSTRUCTION ADDRESS register and the address of the sixteenth level of the defer loop is stored in the INVALID ADDRESS register. The MMPU then disables I/O interrupt requests, enters the supervisor mode, and directs the CPU to fetch the next instruction from physical location 41 (octal).

Write Protection

If write-protection is enabled, the MMPU monitors all modify memory requests and determines whether or not that logical page is write-protected. If the page is not write-protected, the MMPU allows the operation to proceed. If the page is write-protected, the MMPU stores the instruction address in the INSTRUCTION ADDRESS register and stores the memory address in the INVALID ADDRESS register. The MMPU then disables I/O interrupt requests, enters the supervisor mode, and directs the CPU to fetch the next instruction from physical location 41 (octal).

Device Interrupt Processing

Because of the way in which the MMPU disables I/O interrupt requests upon entry to a trap routine, the supervisor should execute an INTDS instruction as soon as possible in the trap routine. If the supervisor does not issue this INTDS instruction, then upon issuing the INTEN instruction, the interrupt system is enabled immediately, not after one more fetch or defer cycle. This means that it is possible for an interrupt service routine to begin executing in user mode.

Example:

```
                    ;ENTRY TO TRAP ROUTINE
        .
        .
        .
                    ;NO INTDS INSTRUCTION
        .
        .
        .
        NIOS   2
        INTEN       First interrupt could
        JMP    @.+1 occur here
        ADDR        ;USER START ADDRESS
```

The installation of the MMPU causes a small change in the normal device interrupt procedure. Normally, when the CPU processes a device interrupt, the Program Counter (PC) is stored in physical location 0 and the CPU does a jump indirect to physical location 1. With the MMPU installed, the PC is stored in logical location 0, the MMPU is placed in supervisor mode, and the CPU does a jump indirect to physical location 1. This is done so that the supervisor's job of restarting the user after handling the interrupt will be simplified.

ADDRESS TRANSLATION USING THE MAP FEATURE

The memory location and protection (MAP) feature available with the SUPERNOVA, NOVA 800, and NOVA 820 computers also performs address translation. The main differences between the MAP feature and the MMPU feature are that the MAP feature does not expand the physical address space and the MAP feature does not translate addresses for the data channel. The MAP feature does, however, provide information on which areas of memory the user has modified. This simplifies the task of swapping.

With the MAP feature, each user has a memory area allocated to him and he cannot gain access to the rest of memory for either storage or retrieval of information. Moreover, part of his allocated area may be write-protected. The supervisor would do this when part of the allocated area contains a pure procedure to be used reentrantly by several users. While the MAP feature is in user mode, the program is further restricted in that it is illegal to issue any I/O instruction except MULTIPLY and DIVIDE. It is also illegal to use more than two levels of indirect addressing. The violation of any restriction by a user program causes the processor to terminate the instruction immediately and return control to the supervisor by initiating a program interrupt request.

For allocation purposes, physical memory is divided into blocks of 4096 words each, defined by the three high-order address bits. For each user, the supervisor establishes a map of the logical blocks used by the program into the physical blocks of memory and validates those logical blocks that are available to the given user.

For protection purposes, physical memory is divided into pages of 256 words each. The supervisor establishes a protection scheme for all of this memory, and although a given user can access any location in his allocated blocks, he cannot write in any page that is protected. To save swapping time, a Page Written flag is associated with each 256-word page. When setting up a user program, the supervisor should clear all the flags. Whenever the user writes in a given page, its associated Page Written flag is set. Then, when that user goes on the inactive list, the supervisor need rewrite on the swapping disc only those pages that have actually changed.

Note that these restrictions apply only to the user program. Data channel transfers can occur while the processor is in user mode, and access is made to the physical locations addressed. An interrupt always returns the MAP feature to supervisor mode--the supervisor handles all interrupts.

Map Feature Instruction Set

The MAP feature is programmed with 19 I/O instructions. Through the use of these instructions, the supervisor tells the MAP feature what the address translation function is to be, tells the MAP feature what pages are to be write-protected, and interrogates the MAP feature as to the status of the current user program.

The instructions for the MAP feature are in the standard I/O format. The device codes for the MAP feature are 2, 3, and 4. The MAP feature responds with device code 2 to the INTERRUPT ACKNOWLEDGE instruction.

At power turnon, the MAP feature is in supervisor mode and the mapping and protection data is indeterminate. The I/O reset switch places the MAP feature in supervisor mode, but does not affect the mapping and protection data.

ASSIGN LOWER LOGICAL MEMORY MAP

DOB <u>ac</u>,MAP0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MAP feature. The contents of the specified AC remain unchanged. The format of the specified AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOGICAL BLOCK 3				LOGICAL BLOCK 2				LOGICAL BLOCK 1				LOGICAL BLOCK 0			

In each set of four bits the leftmost bit is the validity bit. A 1 in this bit makes references to that logical block valid. A 0 in this bit makes references to that logical block invalid. The remaining three bits specify the high-order three bits of the physical memory block to which addresses in the corresponding logical block will be mapped.

ASSIGN UPPER LOGICAL MEMORY MAP

DOC <u>ac</u>,MAP0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MAP feature. The contents of the specified AC remain unchanged. The format of the specified AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOGICAL BLOCK 7				LOGICAL BLOCK 6				LOGICAL BLOCK 5				LOGICAL BLOCK 4			

In each set of four bits the leftmost bit is the validity bit. A 1 in this bit makes references to that logical block valid. A 0 in this bit makes references to that logical block invalid. The remaining three bits specify the high-order three bits of the physical memory block to which addresses in the corresponding logical block will be mapped.

WRITE PROTECT

DOA <u>ac</u>,MAP0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	0	0	0	0	0	1	0

The contents of the specified AC are transferred to the MAP feature. The contents of the specified AC remain unchanged. The format of the specified AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PROTECT PAGES												PHYSICAL HALF BLOCK			
7	6	5	4	3	2	1	0								

Bits 12-15 specify the high-order four bits of a 2048-word half-block. Bits 0-7 select the eight 256-word pages within this half-block. A 1 in any of bits 0-7 protects the corresponding page from being altered by the user. A 0 in any of bits 0-7 permits alteration. The first page in a half-block is page 0, selected by bit 7.

READ STATUS

DIA <u>ac</u>,MAP1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	0	0	0	0	0	0	1	1

The status register of the MAP feature is placed in the specified AC. The previous contents of the specified AC are lost. The format of the status register is as follows:

0	1-8	9	10	11	12	13	14	15
USER		INDIRECT ERROR	IO ERROR	VALIDITY ERROR	PROTECT. ERROR	PHYSICAL BLOCK ADDRESSED		

BITS	CONTENTS
0	User Mode. The last interrupt occurred while in user mode.
1-8	Unused.
9	Indirect Error. The last user instruction attempted more than two levels of indirection.
10	I/O Error. The last user instruction was an I/O instruction other than MULTIPLY or DIVIDE.
11	Validity Error. The last user instruction attempted to reference an invalid logical address.
12	Protection Error. The last user instruction attempted to write in a write-protected page.
13-15	Physical block. These are the high-order three bits of the last address mapped.

While the MAP feature is in user mode, the setting of any of bits 9-12 by the MAP feature immediately causes the MAP feature to enter the supervisor mode and initiate a program interrupt request.

SELECT MODE

DOA<f> ac,MAP1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	F		0	0	0	0	1	1

Bit 0 of the specified AC is placed in bit 0 of the status register. Bits 1-15 of the status register are set to 0. The contents of the specified AC remain unchanged. After the data transfer is complete, the function specified by F is performed.

If a Start command is issued with this instruction (DOAS ac,MAP1) then the Interrupt On flag is set to 1 and the MAP feature is placed in the mode specified by bit 0 of the status register. If this bit is 1, one more instruction will be executed before entering user mode. If this bit is 0, the MAP feature will remain in supervisor mode. If the Interrupt On flag was 0 before being set to 1 by this instruction, one more instruction will be executed before the first interrupt can start.

ENTER USER MODE

NIOS MAP1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	1

The Interrupt On flag is set to 1 and the MAP feature is placed in the mode specified by bit 0 of the status register. If this bit is 1, one more instruction will be executed before entering user mode. If this bit is 0, the MAP feature will remain in supervisor mode. If the Interrupt On flag was 0 before being set to 1 by this instruction, one more instruction will be executed before the first interrupt can start.

MAP AN ADDRESS

DOB ac,MAP1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	0	0	0	0	0	1	1

The logical address contained in bits 1-15 of the specified AC is mapped as if it were a write reference. Any violation will be indicated in the status register. The contents of the specified AC remain unchanged.

READ MAPPED ADDRESS

DIB ac,MAP1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	1	0	0	0	0	0	0	1	1

The physical address which is the result of the last MAP AN ADDRESS instruction is placed in bits 1-15 of the specified AC. Bit 0 of the specified AC is set to 0. The previous contents of the specified AC are lost.

SELECT PAGE WRITTEN CHECK

DOB ac,MAP2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	0	0	0	0	1	0	0

The 4096 word block of physical memory specified by bits 12-15 of the specified AC is selected for page written checking. If bit 15 of the specified AC is 1, the block selected consists of the upper 2048 words of the physical block specified by bits 12-14 of the specified AC and the lower 2048 words of the next higher physical block. The contents of the specified AC remain unchanged.

READ PAGE WRITTEN FLAGS

DIA<f> ac,MAP2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	F		0	0	0	1	0	0

The page written flags associated with the currently selected 4096 block of physical memory are placed in the specified AC. The previous contents of the specified AC are lost. After the data transfer, the function specified by F is performed.

If a Pulse command is issued with this instruction (DIAP) then the next higher pair of contiguous half-blocks is selected for page written checking. If no command function is given then the next 4096 word block selected for page written checking consists of the upper half of the current block and the next contiguous 2048 half-block.

The format of the specified AC is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PAGES WRITTEN IN NEXT HALF BLOCK								PAGES WRITTEN IN SPECIFIED HALF BLOCK							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

CLEAR PAGE WRITTEN FLAGS

DOA 0,MAP2

0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

All the page written flags are set to 0. Physical block 0 is selected for page written checking. The contents of AC0 remain unchanged.

SKIP IF ANY VIOLATION

SKPDN MAP0

0	1	1	0	0	1	1	1	1	0	0	0	0	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If any of bits 9-12 of the status register are set to 1, the next sequential instruction is skipped.

SKIP IF NO VIOLATION

SKPDZ MAP0

0	1	1	0	0	1	1	1	1	1	0	0	0	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If none of bits 9-12 of the status register are set to 1, the next sequential instruction is skipped.

SKIP IF I/O VIOLATION

SKPBN MAP0

0	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 10 of the status register is set to 1, the next sequential instruction is skipped.

SKIP IF NO I/O VIOLATION

SKPBZ MAP0

0	1	1	0	0	1	1	1	0	1	0	0	0	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 10 of the status register is set to 0, the next sequential instruction is skipped.

SKIP IF VALIDITY VIOLATION

SKPDN MAP1

0	0	1	0	0	1	1	1	1	0	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 11 of the status register is set to 1, the next sequential instruction is skipped.

SKIP IF NO VALIDITY VIOLATION

SKPDZ MAP1

0	1	1	0	0	1	1	1	1	1	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 11 of the status register is set to 0, the next sequential instruction is skipped.

SKIP IF PROTECTION VIOLATION

SKPBN MAP1

0	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 12 of the status register is set to 1, the next sequential instruction is skipped.

SKIP IF NO PROTECTION VIOLATION

SKPBZ MAP1

0	1	1	0	0	1	1	1	0	1	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If bit 12 of the status register is set to 0, the next sequential instruction is skipped.

FLOATING POINT ARITHMETIC

In addition to performing fixed point arithmetic, computers in the NOVA line can perform floating point arithmetic if they are equipped with the floating point unit. This feature provides the capability to perform rapid and convenient arithmetic operations on numbers with a much larger range than would be feasible using the fixed point arithmetic instruction set. The precision with which these numbers can be manipulated exceeds the precision readily available with the fixed point instruction set.

Floating Point Unit Registers

There are three registers available to the programmer in the Floating Point Unit (FPU).

These are the Floating Point accumulator (FPAC), the Status Register (SR), and the Temporary Buffer (TEMP). FPAC and TEMP are used for computations and SR is used to control and monitor the operation of the FPU.

FPAC and TEMP can both contain either single or double precision floating point numbers. SR is a 16-bit register containing bits that reflect the current status of FPAC and the FPU. The format of SR is as follows:

STATUS BITS	RESERVED	MODE BITS
0 1 2 3 4 5 6 7	8 9 10 11 12	13 14 15

STATUS REGISTER BITS

BIT	MNEMONIC	MEANING WHEN SET
0	ANY	Indicates that any of bits 1-4 are set.
1	OVF	Overflow indicator meaning that during processing of an FPU instruction, the FPU detected an exponent overflow. The result is correct except that the exponent is 128 too small.
2	UNF	Underflow indicator meaning that during processing of an FPU instruction, the FPU detected an exponent underflow. The result is correct except that the exponent is 128 too large.
3	DVZ	During a divide instruction, the FPU has detected a zero divisor. The division was aborted and FPAC remains unchanged.
4	MOF	Mantissa overflow indicator meaning that during a scale instruction, a left shift was required.
5	GTZ	Greater than indicator, meaning that the operand in FPAC is positive and the mantissa is different from zero.
6	EQZ	Equal indicator, meaning that the operand in FPAC is equal to true zero. This bit examines only the mantissa and sign of FPAC.
7	LTZ	Less than indicator, meaning that the operand in FPAC is less than zero.
8-12		Reserved for future use.
13	IND	Interrupt Disable bit means that the FPU will not interrupt the program for an exponent overflow, exponent underflow, or divide by zero.
14	PPM	Parallel processing mode means that the FPU will not request data channel cycles for the entire time it is processing an instruction. Therefore, the programmer must check the BUSY status of the FPU before issuing the next FPU instruction.
15	DMD	Diagnostic mode means that the program can issue clock pulses and monitor the progress of the FPU cycle by cycle. The data channel will not be held during this mode.

INSTRUCTION SET

Because the FPU is considered an I/O device by the CPU, FPU instructions are really I/O instructions and take the I/O format. The device codes for the FPU are as follows:

MNEMONIC	DEVICE CODE	MEANING
FPU1	74_8	Floating Point-Single Precision
FPU2	75_8	Floating Point-Double Precision
FPU	76_8	Floating Point Unit-used for status instructions and in diagnostic mode.

The programmer can either write I/O instructions for the FPU, or he can use the .DUSR and .DIAC functions of the assembler and define his own mnemonics. A paper tape containing .DUSR and .DIAC functions describing the DGC standard floating point mnemonics is supplied with the FPU. A detailed discussion of this tape can be found under Floating Point Unit Mnemonics. In describing the instructions available for the FPU, both the I/O instruction and the corresponding DGC mnemonic will be shown. For a further discussion of I/O instructions in general, see the I/O section of this manual.

When processing a floating point instruction, the FPU assumes the following:

1. In instructions that refer to operands in memory, the accumulator specified by AC is assumed to contain the address of the first word of the storage that contains or will receive a floating point number. This area is either 2 or 4 words long, depending on the precision specified.
2. In instructions that refer to an operand coming from memory, the number is assumed to be in the format described under "Number Representation". The number is assumed to be normalized.
3. In arithmetic instructions, it is assumed that a floating point number is already present in FPAC.

LOAD SINGLE

.FLDS ac

DOB ac,FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	1	1	1	1	1	1	0	0

LOAD DOUBLE

.FLDD ac

DOBP ac,FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	1	1	1	1	1	1	0	1

The FPAC is loaded with the floating point number contained in storage starting with the address in the specified AC. The operation proceeds one word at a time, starting with the most significant word. Two words are transferred for single precision. Four words are transferred for double precision. The operand in storage and the address in the specified AC remain unchanged. For single precision, the 32-bit floating point number goes into the high-order 32 bits of FPAC and the low-order 32 bits of FPAC are set to zero.

STORE SINGLE

.FSRS ac

DOBS ac,FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	1	1	1	1	1	0	0

STORE DOUBLE

.FSRD ac

DOBS ac,FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	1	1	1	1	1	0	1

The FPAC is stored into memory starting at the address contained in the specified AC. The operation proceeds one word at a time, starting with the most significant word. Two words are transferred for single precision. Four words are transferred for double precision. The number in FPAC and the address in the specified AC remain unchanged.

ADD SINGLE

.FAS ac

DOA ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	0	1	1	1	1	0	0

ADD DOUBLE

.FAD ac

DOA ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	0	1	1	1	1	0	1

The floating point number which starts at the address contained in the specified AC is added to the floating point number in the FPAC. The result is normalized and remains in the FPAC. The operand in storage is transferred to the FPU, most significant word first, before the add operation takes place. Two words are transferred for single precision. Four words are transferred for double precision. The operand in storage and the address in the specified AC remain unchanged. For single precision, the low-order 32 bits of the FPAC are turned to zero before the operation.

Floating point addition consists of an exponent comparison and a mantissa addition. The exponents of the two numbers are compared, and the mantissa of the number with the smaller exponent is shifted right. This exponent alignment is accomplished by taking the absolute value of the difference between the two exponents and shifting the mantissa right that number of hex digits. For double precision, bits shifted out of the right end of the mantissa are lost, and do not take part in the addition. For single precision, the last 8 bits shifted out are retained as hex "guard" digits. This increases the accuracy of single precision addition. If all significant digits are shifted out of the mantissa, the operation is equivalent to adding the number with the larger exponent to zero. This requires a shift of at least 8 hex digits in single precision and at least 14 hex digits in double precision.

After alignment, the FPU adds the mantissas together. The result of this addition is termed the intermediate result. The sign of the result is determined from the sings of the two operands by the rules of algebra. If the mantissa addition produced a carry out of the high-order bit, the mantissa in the intermediate result is shifted right one hex digit and the exponent is incremented by one. If this shift produces an exponent overflow, the OVF bit is set in the SR, and the instruction is terminated. When this condition occurs, the number in the FPAC is correct except that the exponent is 128 too small.

If there is no overflow, the mantissa of the intermediate result is examined for leading hex zeroes. If the mantissa is found to be all zeroes, a true zero is placed in the FPAC and the instruction is terminated.

If the mantissa is non-zero, the intermediate result is normalized, and the number placed in FPAC. If the normalization results in an exponent underflow, the UNF bit is set in the SR and the instruction is terminated. The number in the FPAC is correct except that the exponent is 128 too large.

Upon termination, the FPU sets the appropriate condition code bits in the SR.

SUBTRACT SINGLE

.FSS ac

DOAS ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	1	1	1	1	1	0	0

SUBTRACT DOUBLE

.FSD ac

DOAS ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	1	1	1	1	1	0	1

The floating point number which starts at the address contained in the specified AC is subtracted from the floating point number in the FPAC. The result is normalized and remains in the FPAC. The operand in storage is transferred to the FPU, most significant word first, before the subtract operation takes place. Two words are transferred for single precision. Four words are transferred for double precision. The operand in storage and the address in the specified AC remain unchanged.

Before the operation takes place, the sign bit of the operand fetched from storage is inverted. After the inversion, the operation is equivalent to addition.

MULTIPLY SINGLE

.FMS ac

DOAP ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	1	1	1	1	1	1	0	0

MULTIPLY DOUBLE

.FMD ac

DOAP ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	1	1	1	1	1	1	0	1

The floating point number in the FPAC is multiplied by the floating point number which starts at the address contained in the specified AC. The result is normalized and remains in the FPAC. The operand in storage is transferred to the FPU, most significant word first, before the multiply operation takes place. Two words are transferred for single precision. Four words are transferred for double precision. The operand in storage and the address in the specified AC remain unchanged.

For single precision, the low-order 32 bits of the FPAC are ignored during the operation and are zeroed in the result.

The mantissas of the two numbers are multiplied together to give the mantissa of the intermediate result. The exponents of the two numbers are added together and 64 is subtracted. This subtraction of 64 maintains the "Excess 64" notation. The result of the exponent manipulation becomes the exponent of the intermediate result. The sign of the intermediate result is determined from the signs of the two operands by the rules of algebra.

If the exponent processing produces either overflow or underflow, the result is held until normalization, as that procedure may correct the condition. If normalization does not correct the condition, the corresponding bit in the SR is set. The number in the FPAC is correct except that, for exponent overflow, the exponent is 128 too small, and for exponent underflow, the exponent is 128 too large.

DIVIDE SINGLE

.FDS ac

DOA ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	1	0	1	1	1	1	0	0

DIVIDE DOUBLE

.FDD ac

DOAC ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	1	0	1	1	1	1	0	1

The floating point number in the FPAC is divided by the floating point number which starts at the address contained in the specified AC. The result is normalized and remains in the FPAC. The operand in storage is transferred to the FPU, most significant word first, before the divide operation takes place. Two words are transferred for single precision. Four words are transferred for double precision. The operand in storage and the address in the specified AC remain unchanged.

For single precision, the low-order 32 bits of the FPAC are ignored during the operation and are zeroed in the result.

The operand from storage is checked for a zero mantissa. If the mantissa is zero, the DVZ bit is set in the SR and the instruction is terminated. The number in the FPAC remains unchanged.

The two mantissas are then compared and if the mantissa of the number in the FPAC is greater than or equal to the mantissa of the operand from storage, the mantissa of the number in the FPAC is shifted right one hex digit and the exponent of the number in the FPAC is increased by one. Since all operands are assumed to be normalized, this guarantees that the mantissa of the number in the FPAC will always be less than the mantissa of the operand from storage.

The mantissa in the FPAC is then divided by the mantissa from storage and the quotient is the mantissa of the intermediate result. The exponent from storage is subtracted from the exponent in the FPAC and 64 is added to this result. This addition of 64 maintains the "Excess 64" notation. The result of the exponent manipulation becomes the exponent of the intermediate result. The sign of the intermediate result is determined from the sign of the two operands by the rules of algebra.

If the exponent processing produces either overflow or underflow, the result is held until normalization, as that procedure may correct the condition. If normalization does not correct the condition, the corresponding bit in the SR is set. The number in the FPAC is correct except that, for exponent overflow, the exponent is 128 too small, and for exponent underflow, the exponent is 128 too large.

Temporary Buffer Instructions

The Temporary Buffer, or TEMP, is an area within the FPU capable of holding a single or double precision floating point number. The following instructions make use of this facility.

MOVE FPAC TO TEMP

.FMFT

NIOP FPU2

0	1	1	0	0	0	0	0	1	1	1	1	1	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The double precision floating point number in the FPAC is moved to the TEMP buffer. The number in the FPAC remains unchanged.

MOVE TEMP TO FPAC

.FMTF

NIOC FPU2

0	1	1	0	0	0	0	0	1	0	1	1	1	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The double precision floating point number in the TEMP buffer is moved to the FPAC. The number in the TEMP buffer remains unchanged.

NOTE The operands in these two instructions are 64 bit floating point numbers. If the previous instruction that referred to the FPAC was a single precision instruction, then that instruction zeroed the low-order half of the FPAC and the FPAC can be considered a double precision number with no problem.

ADD TEMP TO FPAC (SINGLE)

. FATS

DOC 0, FPU1

0	1	1	0	0	1	1	0	0	0	1	1	1	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ADD TEMP TO FPAC (DOUBLE)

. FATD

DOC 0, FPU2

0	1	1	0	0	1	1	0	0	0	1	1	1	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The floating point number in TEMP is added to the floating point number in the FPAC and the normalized result is placed in the FPAC. The number in TEMP remains unchanged.

For single precision, only the high-order 32 bits of TEMP and FPAC participate in the operation.

The ADD TEMP TO FPAC instruction is identical to the ADD instruction described previously, except that the second operand comes from TEMP, not from memory.

SUBTRACT TEMP FROM FPAC (SINGLE)

. FSTS

DOC 0, FPU1

0	1	1	0	0	1	1	0	0	1	1	1	1	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

SUBTRACT TEMP FROM FPAC (DOUBLE)

. FSTD

DOCS 0, FPU2

0	1	1	0	0	1	1	0	0	1	1	1	1	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The floating point number in TEMP is subtracted from the floating point number in the FPAC and the normalized result is placed in the FPAC. The number in TEMP remains unchanged.

For single precision, only the high-order 32 bits of TEMP and FPAC participate in the operation.

The SUBTRACT TEMP FROM FPAC instruction is identical to the SUBTRACT instruction described previously, except that the second operand comes from TEMP, not from memory.

MULTIPLY FPAC BY TEMP (SINGLE)

. FMTS

DOCP 0, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	0	1	1	1	1	1	1	0	0

MULTIPLY FPAC BY TEMP (DOUBLE)

. FMTD

DOCP 0, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	0	1	1	1	1	1	1	0	1

The floating point number in the FPAC is multiplied by the floating point number in TEMP and the normalized result is placed in the FPAC. The number in TEMP remains unchanged.

For single precision, only the high-order 32 bits of TEMP and FPAC participate in the operation.

The MULTIPLY FPAC BY TEMP instruction is identical to the MULTIPLY instruction described previously, except that the second operand comes from TEMP not from memory.

DIVIDE FPAC BY TEMP (SINGLE)

. FDTS

DOCC 0, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	0	1	0	1	1	1	1	0	0

DIVIDE FPAC BY TEMP (DOUBLE)

. FDTD

DOCC 0, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	0	1	0	1	1	1	1	0	1

The floating point number in the FPAC is divided by the floating point number in TEMP and the normalized result is placed in the FPAC. The number in TEMP remains unchanged.

For single precision, only the high-order 32 bits of TEMP and FPAC participate in the operation.

The DIVIDE FPAC BY TEMP instruction is identical to the DIVIDE instruction described previously, except that the second operand comes from TEMP not from memory.

Shift and Logical Instructions

The following FPU instructions are included to enable the programmer to convert numbers from integer representation to floating point representation and vice-versa. This section also contains instructions for logical operations and for working with the Status Register.

ABSOLUTE VALUE

.FABS

NIOP FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0

The sign bit of the FPAC is forced to zero. Bits 1-63 of the FPAC remain unchanged.

CLEAR FPAC

.FCLR

NIOS FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0

All 64 bits of the FPAC are forced to zero. In other words, the value of the FPAC is forced to true zero.

LOAD EXPONENT

.FLDX ac

DOBC ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	1	0	1	1	1	1	0	1

Bits 1-7 of the specified AC replace bits 1-7 of the FPAC. Bits 0 and 8-15 of the specified AC are ignored. Bits 0 and 8-63 of the FPAC remain unchanged. The entire contents of the specified AC remain unchanged.

> **NOTE** The exponent is assumed to be in "Excess 64" representation.

NEGATE

.FNEG

NIOC FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	0	1	1	1	1	0	0

The sign bit of the FPAC is inverted. Bits 1-63 of the FPAC remain unchanged.

> **NOTE** If the number in the FPAC is true zero, the sign bit of the FPAC remains zero.

NORMALIZE

.FNRM

NIOS FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	0	1	1	1	1	1	0	1

The floating point number in the FPAC is normalized. If all bits of the mantissa are zero, a true zero is set in the FPAC. If an exponent underflow occurs, the UNF bit in the SR is set and the number is correct, except that the exponent is 128 too large.

READ HIGH WORD

.FHWD ac

DIA ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	0	0	1	1	1	1	0	0

The high-order 16 bits of the FPAC are placed in the specified AC. The previous contents of the specified AC are lost. The contents of the FPAC remain unchanged.

SCALE

.FSCL ac

DOB ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		1	0	0	0	0	1	1	1	1	0	1

The mantissa of the floating point number in the FPAC is shifted either right or left, depending upon the contents of bits 1-7 of the specified AC. The contents of the specified AC remain unchanged.

Bits 1-7 of the specified AC are treated as an exponent in "Excess 64" representation. The difference between this exponent and the FPAC exponent is computed and compared to zero. If the difference is zero, the instruction is terminated. If the difference is positive, the FPAC mantissa is shifted right that number of hex digits. If the difference is negative, the FPAC mantissa is shifted left that number of hex digits and the MOF bit in the SR is set. After the shift, the contents of bits 1-7 of the specified AC replace the FPAC exponent.

Bits shifted out either end of the mantissa are lost.

Status Instructions

READ STATUS

.FRST ac

DIAC ac, FPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	1	0	1	1	1	1	1	0

The contents of the 16 bit status register are placed in the specified AC in the format shown previously. Bits 0-4 of the SR are set to zero.

WRITE STATUS

.FWST ac

DOA ac, FPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	0	0	0	1	1	1	1	1	0

The contents of the specified AC are placed in the status register. The contents of the specified AC remain unchanged.

Diagnostic Instructions

NOTE The following instructions are for diagnostic use only.

READ WORD 1

DIA ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	0	0	1	1	1	1	0	0

READ WORD 2

DIB ac, FPU1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	1	0	0	1	1	1	1	0	0

READ WORD 3

DIA ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	0	1	0	0	1	1	1	1	0	1

READ WORD 4

DIB ac, FPU2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	AC		0	1	1	0	0	1	1	1	1	0	1

These instructions read the four most significant words of the FPU arithmetic unit. When the FPU is idle, these words are words 1-4 of the FPAC. When the FPU is in diagnostic mode, these instructions, along with the FPU CLOCK instruction, allow the program to monitor the output of the FPU arithmetic unit.

FPU CLOCK

NIOP FPU

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	0

When placed in diagnostic mode (bit 15 of the SR) and issued an instruction, the FPU will initiate execution, request the data channel cycles required, and halt. This instruction causes a single clock pulse in the FPU. The results of any arithmetic manipulation can then be monitored by the program by the READ WORD instructions. An IORST will force the FPU to the idle state or if enough FPU CLOCK instructions occur, the FPU will eventually go to the idle state by normal sequence.

NOTE Diagnostic commands are for diagnostic purposes only and are not supported in the Assembler. The user should use the STORE FPAC instruction to retrieve the FPAC.

Mode Settings For The Floating Point Unit

The low-order three bits of the Status Register control the mode in which the FPU operates. The mode can be changed with the WRITE STATUS instruction. Bits 13-15 of the Status Register and the modes that they imply are summarized in the following table.

Status Register Modes

BIT 13	BIT 14	BIT 15	PROCESSING MODE
0	0	0	Normal mode--interrupt enabled
1	0	0	Normal mode-interrupt disabled
0	1	0	Parallel mode--interrupt enabled
1	1	0	Parallel mode--interrupt disabled
X	X	1	Diagnostic mode

Note: X = May be either zero or one.

Normal Mode

The FPU is defined to be in normal mode when bits 14 and 15 of the Status Register are both set to 0. In this mode, the FPU will request data channel cycles whenever it is busy processing an instruction. The FPU should always be assigned a lower DCH priority than any device requiring the data channel while the FPU is busy.

Normal mode imposes the following restrictions on instruction ordering, if the FPU is running with any NOVA line computer other than the NOVA 800 computer or the NOVA 820 computer.

1. FPU instructions must be separated by at least one non-FPU instruction, which must not modify the storage operand of the preceding FPU instructions.

2. The operand of a STORE FPAC instruction cannot be tested immediately after the instruction. At least one machine cycle must elapse.

Examples:

```
LDA    1,PTRX      ;LOAD AC1 WITH
                   ; POINTER TO X
.FLDS  1           ;LOAD X TO FPAC--
                   ; SINGLE PRECISION
.FMS   1           ;MULTIPLY X BY
                   ; ITSELF
```

In this case there is no non-FPU instruction between the LOAD and the MULTIPLY. Results will be unpredictable.

```
LDA    3,PTRX      ;LOAD AC3 WITH
                   ; POINTER TO X
.FLDS  3           ;LOAD X TO FPAC--
                   ; SINGLE PRECISION
STA    3,0,3       ;USE X LOCATIONS
                   ; AS HOLD AREA
.FNRM              ;NORMALIZE X
```

In this case the intervening instruction modifies the location which holds the floating point number X. The number loaded into the FPAC would have, as its high-order 16 bits, the pointer to X.

```
LDA    1,PTRX      ;LOAD AC1 WITH
                   ; POINTER TO X
.FLDS  1           ;LOAD X TO FPAC--
                   ; SINGLE PRECISION
LDA    2,PTRY      ;LOAD AC2 WITH
                   ; POINTER TO Y
LDA    3,PTRES     ;LOAD AC3 WITH
                   ; POINTER TO
                   ; RESULT
.FSTS  3           ;STORE FPAC INTO
                   ; RESULT
LDA    1,RESULT    ;LOAD AC1 WITH
                   ; FIRST WORD OF
                   ; RESULT
```

In this case the last instruction of the example will not produce the desired effect. Because of the restrictions discussed above, RESULT does not hold the sum of X and Y at the time of the LDA instruction. After a floating store, one more instruction cycle must elapse before the receiving area contains the contents of the FPAC.

Parallel Mode

The FPU is defined to be in parallel mode when bit 1 is set to 1 and bit 15 is set to 0. In this mode, the FPU will only request data channel cycles if they are required to fetch or store an operand. After the data channel is released, the CPU is free to process instructions in parallel with the FPU. Before the programmer issues another FPU instruction, however, he must ensure that the FPU has finished processing the previous instruction. This may be accomplished in either of two ways:

1. The number of non-FPU instructions between FPU instructions are of sufficient number to guarantee that the FPU will be idle.
2. The programmer must look at the BUSY flag of the FPU and issue the next instruction when the FPU is not busy.

The advantage of parallel processing is that it allows the programmer to use effectively the time the FPU spends in processing instructions. This time may be used for moving operands, updating pointers, etc.

Example:

```
LDA    0,AOP 1 ;LOAD ADDRESS OF
               ; OP1
.FLDS  0       ;LOAD OP1 TO FPAC--
               ; SINGLE PRECISION
.              ;SOME LIST OF IN-
.              ; STRUCTIONS WHERE
.              ; THE TOTAL EXECU-
               ; TION TIME IS GREAT-
               ; ER THAN THAT OF
               ; .FLDS
LDA    1,AOP 2 ;LOAD ADDRESS OF
               ; OP2
.FMS   1       ;MULTIPLY OP1 BY
               ; OP2--SINGLE PRECI-
               ; SION
SKPBZ  FPU     ;BUSY?
JMP    .-1     ;YES
.FSTS  1       ;NO, STORE RESULT IN
               ; OP1
```

Interrupt Enable and Disable

To provide maximum flexibility, the FPU has an interrupt disable bit in the status register (bit 13), and is maskable via the MASK OUT instruction (bit 5). If both these bits are set to 0, the FPU will signal an interrupt for exponent overflow, exponent underflow, or divide by zero. These conditions are represented by bits 1-3 in the status register. Detailed discussions of these conditions can be found in the section entitled "Floating Point Unit Registers". If either or both of the interrupt disable bits is set to 1, the FPU will not request an interrupt for any of the above conditions, but will set the representative bit in the status register and set bit zero of the status register. These bits will remain set to 1 until cleared by the programmer. If running with interrupt disabled, it is the programmer's responsibility to test the status register periodically in order to detect errors in floating point processing.

NOTE The FPU returns 76_8 as the device code in response to the INTA instruction.

FLOATING POINT UNIT MNEMONICS

To enable implementation of the mnemonics used throughout this manual, a paper tape (DGC Part Number 090-001248) is supplied with each floating point unit. This tape is in assembler-readable format and contains .DIAC and .DUSR instructions which define the mnemonics. There are two ways to use this tape, depending on whether or not the user has a supervisor for his machine.

If the user's machine has no supervisor, then he should read this tape into pass 1 of the assembler, then read in his program. After the tape is read into pass 1 of the assembler, the assembler will correctly assemble all mnemonics used in this manual. If the programmer plans on extensive use of these mnemonics, it is advisable that he read in this paper tape to pass 1 of his assembler and then punch out this new version of the assembler. This punched copy of the assembler will always understand the floating point mnemonics.

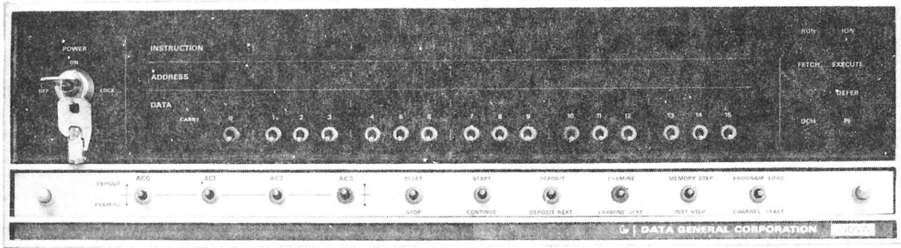
If the user's machine has a supervisor, either DOS or RDOS, then this paper tape should be put on disc as a symbolic file and then specified (with /S switch) as the first file in a multi-file assembly. If this tape is not specified as the first file, floating point mnemonics read into the assembler before this tape is read in, will be flagged as errors.

A table of these .DUSR and .DIAC instructions follows.

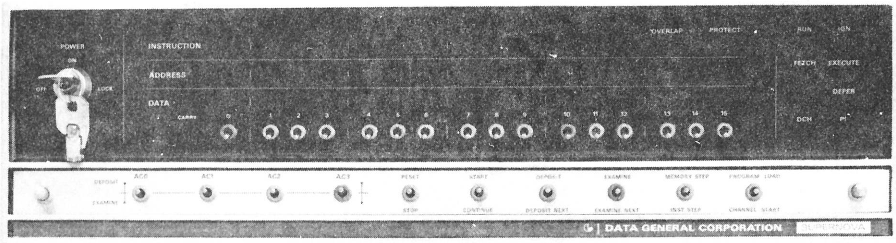
.DUSR and .DIAC Instructions for Floating Point Unit Mnemonics

DEVICE CODES			
.DUSR	FPU=	76	;FLOATING POINT PRIMARY CONTROL
.DUSR	FPU1=	74	;FLOATING POINT SINGLE PRECISION
.DUSR	FPU2=	75	;FLOATING POINT DOUBLE PRECISION
MEMORY REFERENCE INSTRUCTIONS			
.DIAC	.FLDS=	DOBP 0, FPU1	;LOAD SINGLE
.DIAC	.FLDD=	DOBP 0, FPU2	;LOAD DOUBLE
.DIAC	.FSRS=	DOBS 0, FPU1	;STORE SINGLE
.DIAC	.FSRD=	DOBS 0, FPU2	;STORE DOUBLE
ARITHMETIC INSTRUCTIONS			
.DIAC	.FAS=	DOA 0, FPU1	;ADD SINGLE
.DIAC	.FAD=	DOA 0, FPU2	;ADD DOUBLE
.DIAC	.FSS=	DOAS 0, FPU1	;SUBTRACT SINGLE
.DIAC	.FSD=	DOAS 0, FPU2	;SUBTRACT DOUBLE
.DIAC	.FMS=	DOAP 0, FPU1	;MULTIPLY SINGLE
.DIAC	.FMD=	DOAP 0, FPU2	;MULTIPLY DOUBLE
.DIAC	.FDS=	DOAC 0, FPU1	;DIVIDE SINGLE
.DIAC	.FDD=	DOAC 0, FPU2	;DIVIDE DOUBLE
TEMP INSTRUCTIONS			
.DUSR	.FMFT=	NIOP FPU2	;MOVE FPAC TO TEMP
.DUSR	.FMTF=	NIOC FPU2	;MOVE TEMP TO FPAC
.DUSR	.FATS=	DOC 0, FPU1	;ADD TEMP SINGLE
.DUSR	.FATD=	DOC 0, FPU2	;ADD TEMP DOUBLE
.DUSR	.FSTS=	DOCS 0, FPU1	;SUBTRACT TEMP SINGLE
.DUSR	.FSTD=	DOCS 0, FPU2	;SUBTRACT TEMP DOUBLE
.DUSR	.FMTS=	DOCP 0, FPU1	;MULTIPLY TEMP SINGLE
.DUSR	.FMTD=	DOCP 0, FPU2	;MULTIPLY TEMP DOUBLE
.DUSR	.FDTS=	DOCC 0, FPU1	;DIVIDE TEMP SINGLE
.DUSR	.FDTD=	DOCC 0, FPU2	;DIVIDE TEMP DOUBLE
SHIFT AND LOGICAL INSTRUCTIONS			
.DUSR	.FABS=	NIOP FPU1	;ABSOLUTE VALUE
.DUSR	.FCLR=	NIOS FPU1	;CLEAR FPAC
.DIAC	.FLDX=	DOBC 0, FPU2	;LOAD EXPONENT
.DUSR	.FNEG=	NIOC FPU1	;NEGATE
.DUSR	.FNRM=	NIOS FPU2	;NORMALIZE
.DIAC	.FSCL=	DOB 0, FPU2	;SCALE
.DIAC	.FHWD=	DIA 0, FPU1	;READ HIGH WORD
STATUS INSTRUCTIONS			
.DIAC	.FRST=	DIAC 0, FPU	;READ STATUS
.DIAC	.FWST=	DOA 0, FPU	;WRITE STATUS

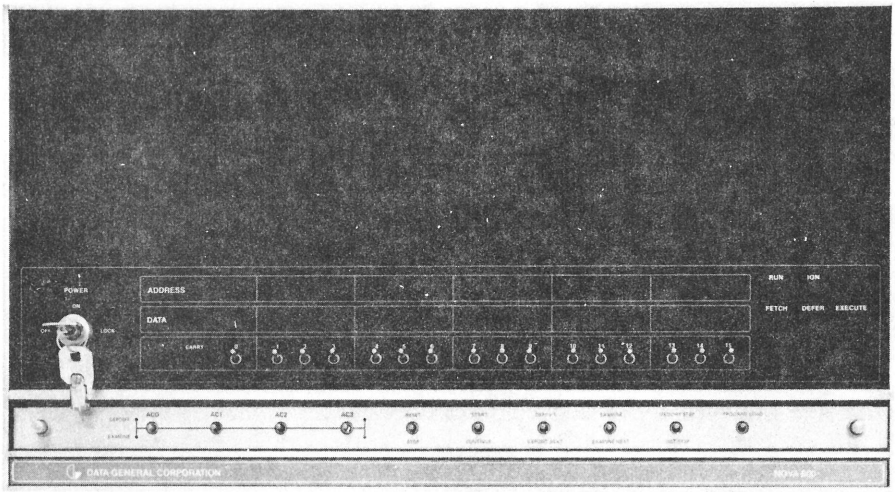
This page intentionally left blank.



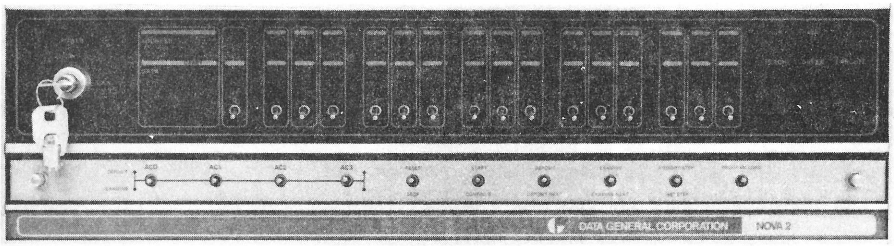
NOVA



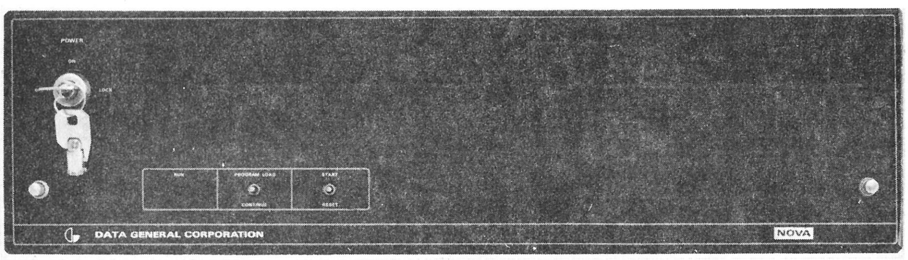
SUPERNOVA



NOVA 800/1200



NOVA 2



TURNKEY

SECTION 6
FRONT PANEL

INTRODUCTION

The front panels of the NOVA line computers contain all the function switches and display all the information needed to operate them. As shown in the figure, all the consoles are essentially the same, except for minor differences in the lights. The console at the top is for the NOVA computer, beneath it is the SUPERNOVA computer console, next is the console found on all NOVA 1200 and NOVA 800 computers, and next is the console for the NOVA 2. The bottom console is a turnkey console, which is available for all NOVA line computers. This console is designed for those computers that will be running in dedicated environments and contains only those switches needed to initiate processing. These switches, and the one light, operate exactly the same as those found on the other consoles.

The function and data switches on the consoles allow the operator to perform many useful operations and the lights reflect the current state of the machine. If a light is lit, it means the corresponding bit is 1. If the light is not lit, the corresponding bit is 0. The lights and their meanings are described below.

FRONT PANEL LIGHTS

LIGHT	MEANING WHEN LIT
ADDRESS	These 15 lights display what is currently in the memory address register.
CARRY	The carry bit is 1.
DATA	These 16 lights display what is currently on the memory bus.
DCH	The next CPU cycle will be used by the data channel to gain access to memory. (NOVA and SUPERNOVA computers only.)
DEFER	The next CPU cycle will be used to follow an indirection chain.
EXECUTE	The next CPU cycle will be used to execute an instruction.
FETCH	The next CPU cycle will be used to fetch an operand or instruction.
INSTRUCTION	These 8 lights display the high-order 8 bits of the instruction just completed. (NOVA and SUPERNOVA computers only.)
ION	The Interrupt On flag is 1.
OVERLAP	Two Accumulator-multiple operation format instructions are being executed out of read-only memory and the CPU is overlapping the execution of one with the fetching of the next. (SUPERNOVA computer only.)
PI	The next CPU cycle will be used to start a program interrupt by storing the program counter in location 0. (NOVA and SUPERNOVA computers only.)
PROTECT	The MAP feature is operating in user mode. (SUPERNOVA computers only.)
RUN	The CPU is executing instructions.

DATA SWITCHES

Beneath the data lights is a row of 16 switches. These switches are used to enter either data or addresses and can be read using the READ SWITCHES instruction. Only switches 1-15 are used for entering addresses. When these switches are in the up position, they represent a 1; when down, they represent a 0.

CONSOLE SWITCHES

Beneath the data switches is a row of 10 function switches. These switches are spring loaded. When pushed up, they perform the function labeled above the switch, and when pushed down, they perform the function labeled below the switch. When released, these switches return to a neutral "off" position. The switches and their functions are explained below.

Accumulator Deposit--Examine

The left-hand four switches reference the four CPU accumulators. The switches are numbered 0-3 from left to right. Each switch affects only its corresponding accumulator. When one of these switches is pushed up, the current setting of the data switches is deposited into the corresponding accumulator. The data lights display the information placed in the AC. When one of these switches is pushed down, the contents of the corresponding accumulator are displayed in the data lights.

Reset--Stop

When this switch is pushed up, the RESET function is performed and an I/O RESET instruction is executed. The CPU is stopped after completing the current processor cycle. The Interrupt On flag, the 16-bit priority mask, and all Busy and Done flags are set to 0.

When this switch is pushed down, the STOP function is performed. The CPU is stopped after completing the current instruction and before executing the next instruction. If an I/O device requests an interrupt during the execution of the current instruction, it is honored before the CPU is stopped. All outstanding data channel requests are honored before the CPU is stopped. After the CPU is stopped, the address lights display the address of the next instruction to be executed, and the data lights display the current contents of the memory bus.

Start--Continue

When this switch is pushed up, the START function is performed. The address indicated by data switches 1-15 is placed in the program counter and sequential operation of the processor begins with the word addressed by the updated value of the program counter.

When this switch is pushed down, the CONTINUE function is performed. Sequential operation of the processor continues from the current state of the machine.

Deposit--Deposit Next

When this switch is pushed up, the DEPOSIT function is performed. The current setting of the data switches is placed into the word addressed by the current value of the program counter. The updated value of the altered word is displayed in the data lights.

When this switch is pushed down, the DEPOSIT NEXT function is performed. The program counter is incremented by one and the current setting of the data switches is placed into the word addressed by the updated value of the program counter. The updated value of the program counter is displayed in the address lights and the updated value of the altered word is displayed in the data lights.

Examine--Examine Next

When this switch is pushed up, the EXAMINE function is performed. The address indicated by data switches 1-15 is placed in the program counter. This value is displayed in the address lights. The contents of the word addressed by the program counter are then read and displayed in the data lights.

When this switch is pushed down, the EXAMINE NEXT function is performed. The current value of the program counter is incremented by one and the new value is displayed in the address lights. The contents of the word addressed by the updated value of the program counter are then read and displayed in the data lights.

Memory Step--Inst Step

When this switch is pushed up, the MEMORY STEP function is performed. The CPU performs a single processor cycle and stops. After the processor stops, the lights indicate the next cycle to be executed.

When this switch is pushed down, the INSTRUCTION SET function is performed. The instruction contained in the word addressed by the current value of the program counter is executed and then the CPU is stopped. The address lights display the updated value of the program counter and the data lights display the contents of the memory bus.

Program Load

In the NOVA 1200, NOVA 800, and NOVA 2 computers, when this switch is pushed up, the PROGRAM LOAD function is performed if the Program Load option is installed on the machine. The contents of the bootstrap read-only memory are placed in memory location 0-37_8 and a "JMP 0" instruction is performed. If the option is not installed, this switch has no effect.

In the SUPERNOVA computer, when this switch is pushed up, the PROGRAM LOAD function is performed. Thirty-three words are read from the device whose device code is set in data switches 10-15 on the console. These words are placed in locations 0-40_8 of main memory. After the last word is read, a "JMP 40" instruction is performed.

Channel Start

When this switch is pushed down, the CHANNEL START function is performed. A "JMP 377" instruction is placed in location 377_8 of main memory. Then a DATA IN A with a Start (DIAS) instruction is issued to the device whose device code is set in data switches 10-15 on the console. After the instruction is issued, a "JMP 377" instruction is performed.

Power

The POWER switch is a three position key switch. The three positions are labeled "OFF", "ON", and "LOCK". With the switch in the OFF position all power to the CPU is shut off and the machine will not run. Turning the switch to the ON position turns on the power and enables all the switches.

Turning the switch to the LOCK position enables the key to be removed. While the CPU is processing and the switch is in the LOCK position, all console functions are disabled. If the switch is turned to the LOCK position while the CPU is stopped or if the CPU executes a HALT instruction while the switch is in the LOCK position, all the function switches are enabled.

PROGRAM LOADING

Before a program can be executed, it must be brought into memory. This requires that a loading program already reside in memory. In the event that there is no loading program in memory, a small, specialized loading program is normally placed in memory and used to read in the loading program. This small loading program is called a "bootstrap loader". The function of the bootstrap loader is to read in a more general-purpose loading program which can be used to load the user's programs. Two methods are available for entering a bootstrap loader into memory. The operator can either enter it via the data switches and the deposit switch, or, if the computer is so equipped, he can use the program load option or the channel start feature.

Manual Loading

When using a NOVA computer or a computer from the NOVA 800, NOVA 1200, or NOVA 2 series without the program load option, a bootstrap loader must be entered into memory manually using the switches on the console. The following loader is the bootstrap loader designed by DGC for use with binary loader #091-000004.

```
LOCATION CONTENTS
  X7757    126440   GET:    SUBO   1,1       ;CLEAR AC1 AND
                                             ; CARRY
  X7760    0636--           SKPDN  --        ;DEVICE BUSY?
  X7761    000777           JMP    .-1       ;YES
  X7762    0605--           DIAS   0,--      ;READ FRAME
                                             ; FROM DEVICE
  X7763    127100           ADDL   1,1       ;SHIFT AC1 LEFT
                                             ; 2 BITS
  X7764    127100           ADDL   1,1       ;SHIFT AC1 LEFT
                                             ; 2 BITS
  X7765    107003           ADD    0,1,SNC   ;ADD IN NEW
                                             ; FRAME
  X7766    000772           JMP    GET+1     ;GET NEW FRAME
  X7767    001400           JMP    0,3       ;FULL WORD--
                                             ; RETURN
  X7770    0601--   BSTRP:  NIOS   --        ;PRIME THE DE-
                                             ; VICE
  X7771    004766           JSR    GET       ;GET A WORD
  X7772    044402           STA    .+2       ;STORE IT
  X7773    004764           JSR    GET       ;GET ANOTHER
                                             ; WORD
```

This loader reads in a specially formatted tape from either the paper tape reader or the reader on the console teletypewriter. This tape has only 4 bits per frame and the loader assembles these frames into complete words. This bootstrap should be placed in memory starting at that location which is 20_8 less than the highest available memory location. In other words, for the "X" in the LOCATION column, substitute a 0 for a 4K system, a 1 for an 8K system, a 2 for a 12K system, and so on. For the dashes in the CONTENTS column, substitute 10_8 if the console teletypewriter is being used, or 12_8 if the paper tape reader is being used. After the bootstrap is entered, start it at location X7770.

Automatic Loading

When using a SUPERNOVA computer, a loading program can be placed in memory by using either the PROGRAM LOAD function or the CHANNEL START function available on the console. The PROGRAM LOAD function reads 66 bytes of data from the device whose device code is set in data switches 10-15. These 66 bytes are compressed into 33 16-bit words and placed in memory locations $0\text{-}40_8$. The first two bytes read are placed in location 0, with the first byte read being placed in bits 0-7, and the second byte read being placed in bits 8-15. This process continues until a word is placed in location 40_8. After a word has been stored into location 40_8, a "JMP 40" instruction is executed.

This sequence is designed to be used with binary loader #091-000041.

Alternatively, when using a SUPERNOVA computer, the CHANNEL START function can be used to bring in a loading program. The CHANNEL START function places a "JMP 377" instruction in location 377_8 and then issues a DIAS instruction to the device whose device code is set in data switches 10-15. After issuing the DIAS instruction, a "JMP 377" instruction is executed. This sequence initiates a data channel transfer from the device to memory beginning at memory location 0. The CPU will continue to execute the "JMP 377" instruction until the data channel places a word in that location. After a word has been placed in location 377_8, it is executed as an instruction. Typically, this word is either a HALT or a JUMP into the data that the data channel has placed in the first 377_8 memory locations.

When using a computer from the NOVA 800, NOVA 1200, or NOVA 2 series with the program load option, a loading program can be placed in memory by using the PROGRAM LOAD function available on the console.

To enter a loader program, the operator must first set up the device that is to be used and set its octal device code into data switches 10-15. If the device is a data channel device, set data switch 0 to 1. If the device is a low-speed device, set data switch 0 to 0. After this is done, push the PROGRAM LOAD switch to the up position. The bootstrap loader will be deposited into memory locations $0\text{-}37_8$ and started at location 0.

The bootstrap loader reads the data switches, sets up its own I/O instructions with the specified device code, and then performs a program load procedure depending upon the state of data switch 0.

If the switch is a 1, the bootstrap loader starts the device for data channel storage beginning at location 0 and then loops at location 377_8 until a data channel transfer places a word into that location.

After a word has been placed in location 377_8, it is executed as an instruction. Typically, this word is either a HALT or a JUMP into the data that the data channel has placed in the first 377_8 memory locations.

If data switch 0 is a 0, the bootstrap loader reads the loader program via programmed I/O. The device must supply 8-bit data bytes, and each pair of bytes is stored as a single word in memory; wherein the first and second bytes read become the left and right halves of the word. To simplify the positioning of the tape in the reader, the bootstrap loader ignores leading null characters. It does not begin storing any words until it reads a non-zero synchronization byte. The first word following this synchronization byte must be the negative of the total number of words to be read, including the first word. The number of words to be read, including the first word may not be greater than 192_{10}. The bootstrap loader stores these words beginning at memory location 100_8. After storing the last word read, it transfers control to that location.

> **NOTE** For proper program loading via the data channel, the device used must be initiated for reading by an I/O RESET followed by an NIOS instruction. In addition, it is up to the device to stop reading after 256 words have been read.

Listed below is the standard 32 word bootstrap loader. This program is capable of loading in either of the manners described above.

BOOTSTRAP LOADER

```
BEG:    IORST            ;RESET ALL I/O
        READS 0          ;READ SWITCHES INTO AC0
        LDA   1,C77      ;GET DEVICE MASK (000077)
        AND   0,1        ;ISOLATE DEVICE CODE
        COM   1,1        ;- DEVICE CODE - 1

LOOP:   ISZ   OP1        ;COUNT DEVICE CODE INTO ALL
        ISZ   OP2        ;I/O INSTRUCTIONS
        ISZ   OP3
        INC   1,1,SZR    ;DONE?
        JMP   LOOP       ;NO, INCREMENT AGAIN

        LDA   2,C377     ;YES, PUT JMP 377 INTO LOCATION 377
        STA   2,377
OP1:    060077           ;START DEVICE: (NIOS 0) - 1
        MOVL  0,0,SZC    ;LOW SPEED DEVICE? (TEST SWITCH 0)
C377:   JMP   377        ;NO, GO TO 377 AND WAIT FOR CHANNEL

LOOP2:  JSR   GET+1      ;GET A FRAME
        MOVC  0,0,SNR    ;IS IT NON-ZERO?
        JMP   LOOP2      ;NO, IGNORE AND GET ANOTHER

LOOP4:  JSR   GET        ;YES, GET FULL WORD
        STA   1,@C77     ;STORE STARTING AT 100 2's COMPLEMENT OF WORD COUNT
                         ; (AUTOINCREMENT)
        ISZ   100        ;COUNT WORD - DONE?
        JMP   LOOP4      ;NO, GET ANOTHER
C77:    JMP   77         ;YES - LOCATION COUNTER AND JUMP TO LAST WORD
GET:    SUBZ  1,1        ;CLEAR AC1, SET CARRY
OP2:
LOOP3:  063577           ;DONE?: (SKPDN 0) - 1
        JMP   LOOP3      ;NO, WAIT
OP3:    060477           ;YES, READ IN AC0: (DIAS 0,0) - 1
        ADDCS 0,1,SNC    ;ADD 2 FRAMES SWAPPED - GOT SECOND?
        JMP   LOOP3      ;NO, GO BACK AFTER IT
        MOVS  1,1        ;YES, SWAP THEM
        JMP   0.3        ;RETURN WITH FULL WORD
        0                ;PADDING
```

This page intentionally left blank.

APPENDICES

- I/O DEVICE CODES AND DATA GENERAL MNEMONICS
- OCTAL AND HEXADECIMAL CONVERSION
- ASCII CHARACTER CODES
- DOUBLE PRECISION ARITHMETIC
- INSTRUCTION USE EXAMPLES
- INSTRUCTION EXECUTION TIMES

APPENDIX A

I/O DEVICE CODES AND DATA GENERAL MNEMONICS

OCTAL DEVICE CODE	MNEMONIC	PRIORITY MASK BIT	DEVICE NAME
00	----	--	Unused
01	MDV	--	Multiply/Divide
02	MMPU	--	Memory Management and Protection Unit
02[2]	MAP0		
03	MAP1	--	Memory Allocation and Protection
04	MAP2		
05			
06	MCAT	12	Multiprocessor adapter transmitter
07	MCAR	12	Multiprocessor adapter receiver
10	TTI	14	TTY input
11	TTO	15	TTY output
12	PTR	11	Paper tape reader
13	PTP	13	Paper tape punch
14	RTC	13	Real-time clock
15	PLT	12	Incremental plotter
16	CDR	10	Card reader
17	LPT	12	Line printer
20	DSK	9	Fixed head disc
21	ADCV	8	A/D converter
22	MTA	10	Magnetic tape
23	DACV	--	D/A converter
24	DCM	0	Data communications multiplexor
25			
26			
27			
30	QTY	14	Asynchronous hardware multiplexor
30	SLA	14	Synchronous line adapter
31[2]	IBM1	13	IBM 360/370 interface
32	IBM2		
33	DKP	7	Moving head disc
34	CAS	10	Cassette tape
34[2]	MX1	11	Multiline asynchronous controller
35	MX2		
36	IPB	6	Interprocessor bus--half duplex
37	IVT	6	IPB watchdog timer
40	DPI	8	IPB full duplex input
41	DPO	8	IPB full duplex output
40[3]	SCR	8	Synchronous communication receiver
41[4]	SCT	8	Synchronous communication transmitter
42	DIO	7	Digital I/O
43	DIOT	6	Digital I/O timer

[2] Code returned by INTA

[3] Can be set up with any unused even device code equal to 40 or above

[4] Can be set up with any unused odd device code equal to 41 or above

APPENDIX A (Continued)

I/O DEVICE CODES AND DATA GENERAL MNEMONICS

OCTAL DEVICE CODE	MNEMONIC	PRIORITY MASK BIT	DEVICE NAME
44	MXM	12	Modem control for MX1/MX2
45			
46	MCAT1	12	Second multiprocessor transmitter
47	MCAR1	12	Second multiprocessor receiver
50	MMI1	14	Second TTY input
51	TTO1	15	Second TTY output
52	PTR1	11	Second paper tape reader
53	PTP1	13	Second paper tape punch
54	RTC1	13	Second real-time clock
55	PLT1	12	Second incremental plotter
56	CDR1	10	Second card reader
57	LPT1	12	Second line printer
60	DSK1	9	Second fixed head disc
61	ADCV1	8	Second A/D converter
62	MTA1	10	Second magnetic tape
63	DACV1	--	Second D/A converter
64[2]	FPU1		
65	FPU2	5	Alternate location for floating point
66	FPU		
67			
70	QTY1	14	Second asynchronous hardware multiplexor
70	SLA1	14	Second synchronous line adapter
71[2]			
72		13	Second IBM 360/370 interface
73	DKP1	7	Second moving head disc
74	CAS1	10	Second cassette tape
74[2]			
75		11	Second multiline asynchronous controller
74[2]	FPU1		
75	FPU2	5	Floating point
76	FPU		
77	CPU	--	Central processor and console functions

[2]Code returned by INTA

[3]Can be set up with any unused even device code equal to 40 or above

[4]Can be set up with any unused odd device code equal to 41 or above

This page intentionally left blank

APPENDIX B

OCTAL AND HEXADECIMAL CONVERSION

To convert a number from octal or hexadecimal to decimal, locate in each column of the appropriate table the decimal equivalent for the octal or hex digit in that position. Add the decimal equivalents to obtain the decimal number

To convert a decimal number to octal or hexadecimal:

1. Locate the largest decimal value in the appropriate table that will fit into the decimal number to be converted;
2. note its octal or hex equivalent and column position;
3. find the decimal remainder.

Repeat the process on each remainder. When the remainder is 0, all digits will have been generated.

	8^5	8^4	8^3	8^2	8^1	8^0
0	0	0	0	0	0	0
1	32,768	4,096	512	64	8	1
2	65,536	8,192	1,024	128	16	2
3	98,304	12,228	1,536	192	24	3
4	131,072	16,384	2,048	256	32	4
5	163,840	20,480	2,560	320	40	5
6	196,608	24,576	3,072	384	48	6
7	229,376	28,672	3,584	448	56	7

	16^5	16^4	16^3	16^2	16^1	16^0
0	0	0	0	0	0	0
1	1,048,576	65,536	4,096	256	16	1
2	2,097,152	131,072	8,192	512	32	2
3	3,145,728	196,608	12,288	768	48	3
4	4,194,304	262,144	16,384	1,024	64	4
5	5,242,880	327,680	20,480	1,280	80	5
6	6,291,456	393,216	24,576	1,536	96	6
7	7,340,032	458,752	28,672	1,792	112	7
8	8,388,608	524,288	32,768	2,048	128	8
9	9,437,184	589,824	36,864	2,304	144	9
A	10,485,760	655,360	40,960	2,560	160	10
B	11,534,336	720,896	45,056	2,816	176	11
C	12,582,912	786,432	49,152	3,072	192	12
D	13,631,488	851,968	53,248	3,328	208	13
E	14,680,064	917,504	57,344	3,584	224	14
F	15,728,640	983,040	61,440	3,840	240	15

This page intentionally left blank.

APPENDIX C

ASCII CHARACTER CODES

Decimal	7-bit Octal	ASCII Character	Control Function	To Produce On TTY Mod 33, 35: Cntrl	Shift	Char	Even Parity 8-bit code
0	000	NUL	Null	✓	✓	P	000
1	001	SOH	Start of Heading	✓		A	201
2	002	STX	Start of Text	✓		B	202
3	003	ETX	End of Text	✓		C	003
4	004	EOT	End of Transmission	✓		D	204
5	005	ENQ	Enquiry	✓		E	005
6	006	ACK	Acknowledge	✓		F	006
7	007	BEL	Bell	✓		G	207
8	010	BS	Backspace	✓		H	210
9	011	HT	Horizontal Tab	✓		I	011
10	012	NL	New Line			line feed	012
				✓		J	012
				✓		line feed	212[1]
11	013	VT	Vertical Tab	✓		K	213
12	014	FF	Form Feed	✓		L	014
13	015	RT	Return			return	215
				✓		M	215
				✓		return	015[1]
14	016	SO	Shift Out	✓		N	216
15	017	SI	Shift In	✓		O	017
16	020	DLE	Data Link Escape	✓		P	220
17	021	DC1	Device Control 1	✓		Q	021
18	022	DC2	Device Control 2	✓		R	022
19	023	DC3	Device Control 3	✓		S	223
20	024	DC4	Device Control 4	✓		T	024
21	025	NAK	Negative Acknowledge	✓		U	225
22	026	SYN	Synchronous Idle	✓		V	226
23	027	ETB	End Transmission Block	✓		W	027
24	030	CAN	Cancel	✓		X	030
25	031	EM	End of Medium	✓		Y	231
26	032	SUB	Substitute	✓		Z	232
27	033	ESC	Escape			esc	033
				✓	✓	K	033
28	034	FS	File Separator	✓	✓	L	234
29	035	GS	Group Separator	✓	✓	M	035
30	036	RS	Record Separator	✓	✓	N	036
31	037	US	Unit Separator	✓	✓	O	237
32	040	SP	Space			space	240
33	041	!			✓	1	041
34	042	"			✓	2	042
35	043	#			✓	3	243
36	044	$			✓	4	044
37	045	%			✓	5	245
38	046	&			✓	6	246
39	047	'			✓	7	047
40	050	(			✓	8	050
41	051	)			✓	9	251

[1] On even parity TTY's, these codes are odd parity.

APPENDIX C (Continued)
ASCII CHARACTER CODES

Decimal	7-bit Octal	Character	To Produce On TTY Mod 33, 35 Cntrl Shift Char		Even Parity 8-bit Code
42	052	*	✓	:	252
43	053	+	✓	;	053
44	054	,		,	
45	055	-		-	055
46	056	.		.	056
47	057	/		/	257
48	060	0		0	060
49	061	1		1	261
50	062	2		2	262
51	063	3		3	063
52	064	4		4	264
53	065	5		5	065
54	066	6		6	066
55	067	7		7	267
56	070	8		8	270
57	071	9		9	071
58	072	:		:	072
59	073	;		;	273
60	074	<	✓	,	074
61	075	=	✓	-	275
62	076	>	✓	.	276
63	077	?	✓	/	077
64	100	@	✓	P	300
65	101	A		A	101
66	102	B		B	102
67	103	C		C	303
68	104	D		D	104
69	105	E		E	305
70	106	F		F	306
71	107	G		G	107
72	110	H		H	110
73	111	I		I	311
74	112	J		J	312
75	113	K		K	113
76	114	L		L	314
77	115	M		M	115
78	116	N		N	116
79	117	O		O	317
80	120	P		P	120
81	121	Q		Q	321
82	122	R		R	322
83	123	S		S	123
84	124	T		T	324

APPENDIX C (Continued)
ASCII CHARACTER CODES

Decimal	7-bit Octal	Character	To Produce On TTY Mod 33, 35 Cntrl Shift Char	Even Parity 8-bit Code
85	125	U	U	125
86	126	V	V	126
87	127	W	W	327
88	130	X	X	330
89	131	Y	Y	131
90	132	Z	Z	132
91	133	[	✓ K	333
92	134	\	✓ L	134
93	135	]	✓ M	335
94	136	^	✓ N	336
95	137	_	✓ 0	137
96	140	`		140
97	141	a		341
98	142	b		342
99	143	c		143
100	144	d		344
101	145	e		145
102	146	f		146
103	147	g		347
104	150	h		350
105	151	i		151
106	152	j		152
107	153	k		353
108	154	l		154
109	155	m		355
110	156	n		356
111	157	o		157
112	160	p		360
113	161	q		161
114	162	r		162
115	163	s		363
116	164	t		164
117	165	u		365
118	166	v		366
119	167	w		167
120	170	x		170
121	171	y		371
122	172	z		372
123	173	{		173
124	174	|		374
125	175	}		175
126	176	~		176
127	177	DEL	rubout	377

This page intentionally left blank.

APPENDIX D

DOUBLE PRECISION ARITHMETIC

A double length number consists of two words concatenated into a 32-bit string wherein bit 0 is the sign and bits 1-31 are the magnitude in two's complement notation. The high-order part of a negative number is therefore in one's complement form unless the low-order part is null (at the right only 0's are null regardless of sign). Hence, in processing double length numbers, two's complement operations are usually confined to the low-order parts, whereas one's complement operations are generally required for the high-order parts.

Suppose we wish to negate the double length number whose high and low-order words respectively are in AC0 and AC1. We negate the low-order part, but we simply complement the high-order part unless the low order part is zero. Hence

```
NEG 1,1,SNR
NEG 0,0,SKP ;LOW ORDER ZERO
COM 0,0     ;LOW ORDER NON-ZERO
```

Note that the magnitude parts of the sequence of negative numbers from the most negative toward zero are the positive numbers from zero upward. In other words, the negative representation -x is the sum of x and the most negative number. Hence, in multiple precision arithmetic, low-order words can be treated simply as positive numbers. In unsigned addition a carry indicates that the low-order result is just too large and the high-order part must be increased. We add the number in AC2 and AC3 to the number in AC0 and AC1.

```
ADDZ 3,1,SZC
INC  0,0
ADD  2,0
```

In two's complement subtraction a carry should occur unless the subtrahend is too large. We could increment as in addition, but since incrementing in the high-order part is precisely the difference between a one's complement and a two's complement, we can always manage with only two instructions. We subtract the number in AC2 and AC3 from that in AC0 and AC1.

```
SUBZ 3,1,SZC
SUB  2,0,SKP
ADC  2,0
```

This page intentionally left blank.

APPENDIX E

INSTRUCTION USE EXAMPLES

On the following pages are examples of how the instruction set of the NOVA 2 may be used to perform some common functions.

1. Clear an AC and the carry bit.

```
SUBO     AC,AC
```

2. Clear an AC and preserve the carry bit.

```
SUBC     AC,AC
```

3. Generate the indicated constants.

```
SUBZL    AC,AC          ;GENERATE +1
ADC      AC,AC          ;GENERATE -1
ADCZL    AC,AC          ;GENERATE -2
```

4. Let ACX be any accumulator whose contents are zero. Generate the indicated constants in ACX.

```
INCZL    ACX,ACX        ;GENERATE +2
INCOL    ACX,ACX        ;GENERATE +3
INCS     ACX,ACX        ;GENERATE +400₈
```

5. Subtract 1 from an accumulator without using a constant from memory.

```
NEG      AC,AC
COM      AC,AC
```

6. Check if both bytes in an accumulator are equal.

```
MOVS     ACS,ACD
SUB      ACS,ACD,SZR
JMP      ---            ;NOT EQUAL
---      ---            ;EQUAL
```

7. Check if two accumulators are both zero.

```
MOV      ACS,ACS,SNR
SUB#     ACS,ACD,SZR
JMP      ---            ;NOT BOTH ZERO
---      ---            ;BOTH ZERO
```

8. Check an ASCII character to make sure it is a decimal digit. The character is in ACS and is not destroyed by the test. Accumulators ACX and ACY are destroyed.

```
LDA      ACX,C60        ;ACX=ASCII ZERO
LDA      ACY,C71        ;ACY=ASCII NINE
ADCZ#    ACY,ACS,SNC    ;SKIPS IF (ACS) > 9
ADCZ#    ACS,ACX,SZC    ;SKIPS IF (ACS) ≥ 0
JMP      ---            ;NOT DIGIT
---      ---            ;DIGIT

C60:     60             ;ASCII ZERO
C71:     71             ;ASCII NINE
```

9. Test an accumulator for zero.

```
MOV      AC,AC,SZR
JMP      ---            ;NOT ZERO
---      ---            ;ZERO
```

APPENDIX E (Continued)
INSTRUCTION USE EXAMPLES

10. Test an accumulator for -1.

```
COM#     AC,AC,SZR
JMP      ---            ;NOT -1
---      ---            ;-1
```

11. Test an accumulator for 2 or greater.

```
MOVZR#   AC,AC,SNR
JMP      ---            ;LESS THAN 2
---      ---            ;2 OR GREATER
```

12. Assume it is known that AC contains 0, 1, 2, or 3. Find out which one.

```
MOVZR#   AC,AC,SEZ
JMP      THREE          ;WAS 3
MOV      AC,AC,SNR
JMP      ZERO           ;WAS 0
MOVZR#   AC,AC,SZR
JMP      TWO            ;WAS 2
---      ---            ;WAS 1
```

13. Multiply an AC by the indicated value.

```
MOV      ACX,ACX        ;MULTIPLY BY 1

MOVZL    ACX,ACX        ;MULTIPLY BY 2

MOVZL    ACX,ACY        ;MULTIPLY BY 3
ADD      ACY,ACX

ADDZL    ACX,ACX        ;MULTIPLY BY 4

MOV      ACX,ACY        ;MULTIPLY BY 5
ADDZL    ACX,ACX
ADD      ACY,ACX

MOVZL    ACX,ACY        ;MULTIPLY BY 6
ADDZL    ACY,ACX

MOVZL    ACX,ACY        ;MULTIPLY BY 7
ADDZL    ACY,ACY
SUB      ACX,ACY        ;IN ACY

ADDZL    ACX,ACX        ;MULTIPLY BY 8
MOVZL    ACX,ACX

MOVZL    ACX,ACY        ;MULTIPLY BY 9
ADDZL    ACY,ACY
ADD      ACY,ACX

MOV      ACX,ACY        ;MULTIPLY BY 10 (base 10)
ADDZL    ACX,ACX
ADDZL    ACY,ACX

MOVZL    ACX,ACY        ;MULTIPLY BY 12 (base 10)
ADDZL    ACY,ACX
MOVZL    ACX,ACX

MOVZL    ACX,ACY        ;MULTIPLY BY 18 (base 10)
ADDZL    ACY,ACY
ADDZL    ACY,ACX
```

APPENDIX E (Continued)
INSTRUCTION USE EXAMPLES

14. Perform the inclusive OR of the operands in AC0 and AC1. The result is placed in AC1. The carry bit is unchanged.

```
        COM     0,0
        AND     0,1
        ADC     0,1
```

15 Perform the exclusive OR of the operands in AC0 and AC1. The result is placed in AC1. The contents of AC2 and the carry bit are destroyed.

```
        MOV     1,2
        ANDZL   0,2
        ADD     0,1
        SUB     2,1
```

16. Move 30 words from locations 2000_8 - 2035_8 to locations 3000_8 - 3035_8. Two auto-increment locations are used to hold the source and destination addresses.

```
        LDA     0,ADDRS    ;SET UP SOURCE ADDRESS
        STA     0,20
        LDA     0,ADDRD    ;SET UP DESTINATION ADDRESS
        STA     0,21
LOOP:   LDA     0,@20      ;INCREMENT SOURCE ADDRESS AND GET WORD
        STA     0,@21      ;INCREMENT DESTINATION ADDRESS AND STORE WORD
        DSZ     CNT        ;DECREMENT COUNT
        JMP     LOOP       ;GO BACK FOR NEXT WORD
        ...                ;SKIP HERE WHEN COUNT IS ZERO
        ...
ADDRS:  1777               ;SOURCE ADDRESS MINUS ONE
ADDRD:  2777               ;DESTINATION ADDRESS MINUS ONE
CNT:    36                 ;WORD COUNT--36(8) EQUALS 30(10)
```

17. Perform the following unsigned integer comparisons.

SUB#	ACS, ACD, SZR	;SKIP IF CONTENTS OF ACS	$=$	CONTENTS OF ACD
SUB#	ACS, ACD, SNR	;SKIP IF CONTENTS OF ACS	$\neq$	CONTENTS OF ACD
ADCZ#	ACS, ACD, SNC	;SKIP IF CONTENTS OF ACS	$<$	CONTENTS OF ACD
SUBZ#	ACS, ACD, SNC	;SKIP IF CONTENTS OF ACS	$<$	CONTENTS OF ACD
SUBZ#	ACS, ACD, SZC	;SKIP IF CONTENTS OF ACS	$>$	CONTENTS OF ACD
ADCZ#	ACS, ACD, SZC	SKIP IF CONTENTS OF ACS	$>$	CONTENTS OF ACD

APPENDIX E (Continued)
INSTRUCTION USE EXAMPLES

18. Simulate the operation of the MULTIPLY instruction.

```
.MPYU:   SUBC    0,0        ;CLEAR AC0, DON'T DISTURB CARRY
.MPYA:   STA     3,.CB03    ;SAVE RETURN
         LDA     3,.CB20    ;GET STEP COUNT
.CB99:   MOVR    1,1,SNC    ;CHECK NEXT MULTIPLIER BIT
         MOVR    0,0SKP     ;0 SHIFT

         ADDZR   2,0        ;1 - ADD MULTIPLICAND AND SHIFT
         INC     3,3,SZR    ;COUNT STEP, COMPLEMENTING CARRY ON FINAL COUNT
         JMP     .CB99      ;ITERATE LOOP

         MOVCR   1,1        ;SHIFT IN LAST LOW BIT (WHICH WAS COMPLEMENTED BY
                            ;FINAL COUNT) AND
         JMP     @.CB03     ;RESTORE CARRY

.CB03:   0
.CB20:   -20                ;16(10) STEPS
```

19. Simulate the operation of the DIVIDE instruction.

```
.DIVI:   SUB     0,0        ;INTEGER DIVIDE CLEAR HIGH PART
.DIVU:   STA     3,.CC03    ;SAVE RETURN
         SUB#    2,0,SZC    ;TEST FOR OVERFLOW
         JMP     .CC99      ;YES, EXIT(AC0>AC2)
         LDA     3,.CC20    ;GET STEP COUNT
         MOVZL   1,1        ;SHIFT DIVIDEND LOW PART

.CC98:   MOVL    0,0        ;SHIFT DIVIDEND HIGH PART
         SUB#    2,0,SZC    ;DOES DIVISOR GO IN?
         SUB     2,0        ;YES
         MOVL    1,1        ;SHIFT DIVIDEND LOW PART
         INC     3,3,SZR    ;COUNT STEP
         JMP     CC98       ;ITERATE LOOP
         SUBO    3,3,SKP    ;DONE, CLEAR CARRY
.CC99:   SUBZ    3,3        ;SET CARRY
         JMP     @.CC03     ;RETURN

.CC03:   0
.CC20:   20                 ;16(10) STEPS
```

APPENDIX F

INSTRUCTION EXECUTION TIMES

SUPERNOVA read-only time equals semiconductor time, except add 0.2 for LDA, STA, ISZ, and DSZ if reference is to core. NOVA times are for core; for read-only subtract 0.2 except subtract 0.4 for LDA, STA, ISZ, and DSZ if reference is to read-only memory. When two numbers are given, the one at the left of the slash is the time for an isolated transfer, the one at the right is the minimum time between consecutive transfers. All times are in microseconds.

		SUPERNOVA		1200	800,820		NOVA 2	
	NOVA	SC	CORE	SERIES	840	830	8K	16K
LDA	5.2	1.2	1.6	2.55	1.6	2.0	1.6	2.0
STA	5.5	1.2	1.6	2.55	1.6	2.0	1.6	2.0
ISZ, DSZ	5.2	1.4	1.8	3.15	1.8	2.2	1.7	2.1
JMP	5.6	0.6	0.8	1.35	0.8	1.0	0.8	1.0
JSR	3.5	1.2	1.4	1.35	0.8	1.0	1.1	1.2
COM, NEG, MOV, INC	5.6	0.3	0.8	1.35	0.8	1.0	0.8	1.0
ADC, SUB, ADD, AND	5.9	0.3	0.8	1.35	0.8	1.0	0.8	1.0
Each level of @, add	2.6	0.6	0.8	1.2	0.8	1.0	0.8	1.0
Each autoindex, add	0.0	0.2	0.2	0.6	0.2	0.2	0.5	0.5
Base register addr, add	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
If skip occurs, add	0.0	*	0.8	1.35	0.2	0.2	0.3	0.2
I/O input (except INTA)	4.4	2.8	2.9	2.55	2.2	2.4	1.4	1.5
INTA	4.4	3.6	3.7	2.55	2.2	2.4	1.4	1.5
I/O output	4.7	3.2	3.3	3.15	2.2	2.4	1.6	1.7
NIO	4.4	3.2	3.3	3.15	2.2	2.4	1.6	1.7
I/O skips	4.4	2.8	2.9	2.55	1.4	1.6	1.1	1.2
If skip occurs, add	0.0	0.0	0.0	0.0	0.2	0.2	0.3	0.2
For S, C, or P; add	0.0	0.0	0.0	0.0	0.6	0.6	0.3	0.3
MUL								
Average	11.1	3.7	3.8	3.75	8.8	9.0	6.1	6.2
Maximum	11.1	5.3	5.4	3.75	8.8	9.0	6.1	6.2
DIV								
Successful	11.9	6.8	6.9	4.05	8.8	9.0	6.4	6.5
Unsuccessful	11.9	1.5	1.6	2.55	1.6	2.0	6.4	6.5
P.I. CYCLE	5.2	1.8	2.2	3.0	1.6	2.0	2.2	2.5
INTERRUPT LATENCY								
With MUL/DIV	12.0	9.0	9.0	7.0	10.6	12.0	5.8	5.9
Without MUL/DIV	12.0	5.0	5.0	7.0	4.6	6.0	1.9	2.3
DATA CHANNEL								
Input	3.5	2.3	2.3	1.2	2.0	2.2	2.0	2.1
Output	4.4	2.8	2.8	1.2/1.8	2.0	2.2	2.1	2.2
Increment	4.4	2.8	2.8	1.8/2.4	2.2	2.4	2.2	2.3
Add to memory	5.3	2.8	2.8	----	----	N/A	---	---
Latency+								
With MUL/DIV	17.3	11.8	11.8	9.4	5.8	6.4	5.2	5.3
Without MUL/DIV	17.3	7.8	7.8	9.4	5.8	6.4	5.2	5.3
HIGH SPEED DATA CHANNEL								
Input	N/A	0.8	0.8	N/A	0.8	1.0	0.8	0.9/1.0
Output		0.8/1.0	0.8/1.0		0.8/1.0	1.0/1.2	1.2	1.3
Increment		1.0/1.2	1.0/1.2		1.0/1.2	1.2/1.4	1.3	1.4
Add to memory		1.0/1.2	1.0/1.2	----	----	N/A	---	---
Latency+								
With MUL/DIV		5.7	5.7		4.8	5.4	4.3	4.4
Without MUL/DIV		3.7	3.7		3.2	3.6	4.3	4.4

*If 2AC-multiple operation instruction is skipped, add 0.3; otherwise add 0.6.
+For highest priority peripheral on I/O bus.

APPENDIX F (Continued)

INSTRUCTION EXECUTION TIMES

Floating Point Unit Instruction Execution Times*

FPU INSTRUCTION	BASE TIME (Microseconds)		TOTAL EXECUTION TIME FOR NOVA 800 WITH HIGH SPEED DATA CHANNEL	
	MAXIMUM	MINIMUM	MAXIMUM	MINIMUM
.FLDS	1.2	1.2	6.3	6.3
.FLDD	0.8	0.8	7.9	7.9
.FSRS	0.4	0.4	5.4	5.4
.FSRD	0.0	0.0	7.1	7.1
.FAS	3.8	3.7	8.3	8.2
.FAD	3.4	3.3	9.9	9.8
.FSS	3.8	3.7	8.9	8.8
.FSD	3.4	3.3	10.5	10.4
.FMS	6.9	6.9	12.0	12.0
.FMD	12.9	12.9	20.0	20.0
.FDS	10.1	9.3(2.0)**	15.2	14.4(7.1)**
.FDD	16.1	15.3(1.6)	23.2	22.4(8.7)
.FMFT	1.0	0.9	3.8	3.7
.FMTF	1.0	0.9	3.8	3.7
.FATS	3.6	3.4	5.8	5.6
.FATD	3.6	3.4	5.8	5.6
.FSTS	3.6	3.4	6.4	6.2
.FSTD	3.6	3.4	6.4	6.2
.FMTS	6.7	6.6	9.5	9.4
.FMTD	13.1	13.0	15.9	15.8
.FDTS	9.9	9.0(1.7)**	12.7	11.8(4.5)**
.FDTD	16.3	15.4(1.7)	19.1	18.2(4.5)
.FABS	1.0	0.9	3.8	3.7
.FCLR	1.0	0.9	3.8	3.7
.FLDX	1.0	0.9	3.8	3.7
.FNEG	1.0	0.9	3.8	3.7
.FNRM	1.1	1.0	3.9	3.8
.FSCL	1.1	1.0	3.3	3.2
.FHWD	0.0	0.0	2.2	2.2
.FRST	0.0	0.0	2.8	2.8
.FWST	0.0	0.0	2.2	2.8

*Total Execution time = Base time + I O instruction time + Data Channel time (if any).
**Times in parentheses are times if "divide-by-zero" is sensed.

DG-01365

CUT ALONG DOTTED LINE

READERS COMMENT FORM

DOCUMENT TITLE: ....

Your comments, accompanied by answers to the following questions, help us improve the quality and usefulness of our publications. If your answer to a question is "no" or requires qualification, please explain.

How did you use this publication?

( ) As an introduction to the subject.
( ) As an aid for advanced knowledge.
( ) For information about operating procedures.
( ) To instruct in a class.
( ) As a student in a class.
( ) As a reference manual.
( ) Other..

Did you find the material:

- Useful..........................YES ( ) NO ( )
- Complete.....................YES ( ) NO ( )
- Accurate.....................YES ( ) NO ( )
- Well organized...........YES ( ) NO ( )
- Well written.................YES ( ) NO ( )
- Well illustrated............YES ( ) NO ( )
- Well indexed...............YES ( ) NO ( )
- Easy to read...............YES ( ) NO ( )
- Easy to understand.....YES ( ) NO ( )

We would appreciate any other comments; please label each comment as an addition, deletion, change, or error and reference page numbers where applicable.

COMMENTS

PAGE	COL	PARA	LINE	FROM	TO

From:

NAME.. TITLE..................................
FIRM.. DIV. ....
ADDRESS...
CITY.. STATE.............. ZIP..........
TELEPHONE.............................. DATE......................................

Data General Corporation

ENGINEERING PUBLICATIONS
COMMENT FORM
DG-00935

FOLD DOWN FIRST FOLD DOWN

FIRST CLASS
PERMIT NO. 26
SOUTHBORO
MASS. 01772

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

Postage will be paid by:

DataGeneral
Southboro, Massachusetts 01772

ATTENTION: Engineering Publications

FOLD UP SECOND FOLD UP

STAPLE